FreeBSD® JOURNAL

May/June 2020

TLS Offload in the Kernel

Interview: Michael W Lucas

FreeBSD Desktop Distributions

Bulk Ports Management with Poudriere

TrueNAS® M-SERIES
Powerfully Scalable Enterprise Storage

OPEN STORAGE FOR ENTERPRISE WORKLOADS

UTILIZES FLASH-OPTIMIZED ZFS TECHNOLOGY
IDEAL FOR LATENCY-SENSITIVE AND BUSINESS-CRITICAL VIRTUAL MACHINES AND PHYSICAL WORKLOADS.

- PERFORMANCE AND SCALE WITHOUT COMPROMISE
- INTELLIGENT STORAGE OPTIMIZATION
- SELF-HEALING DATA PROTECTION
- UNLIMITED SNAPSHOTS AND REPLICATION

Contact iXsystems to Learn More about what TrueNAS® can do for your business!

iXsystems.com/TrueNAS | (855) GREP-4-iX

iXsystems™

© 2020 iXsystems. TrueNAS is a registered trademark of iXsystems, Inc. All rights reserved. Intel, the Intel logo, Xeon, and Xeon Inside are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. All product names, logos, and brands are property of their respective owners.

FreeBSD® JOURNAL
Editorial Board

John Baldwin • FreeBSD Developer, Member of the FreeBSD Core Team and Chair of *FreeBSD Journal* Editorial Board.

Bryan Drewery • Senior Software Engineer at Dell EMC Isilon, member of FreeBSD Portmgr Team, and FreeBSD Committer.

Justin Gibbs • Founder of the FreeBSD Foundation, President of the FreeBSD Foundation, and a Software Engineer at Facebook.

Daichi Goto • Director at BSD Consulting Inc. (Tokyo).

Joseph Kong • Software Development Engineer at Amazon Web Services, author of *FreeBSD Device Drivers* and *Designing BSD Rootkits.*

Dru Lavigne • Author of *BSD Hacks* and *The Best of FreeBSD Basics.*

Michael W Lucas • Author of *Absolute FreeBSD.*

Ed Maste • Director of Project Development, FreeBSD Foundation.

Kirk McKusick • Treasurer of the FreeBSD Foundation Board, and lead author of *The Design and Implementation* book series.

George V. Neville-Neil • Director of the FreeBSD Foundation Board, and co-author of *The Design and Implementation of the FreeBSD Operating System.*

Philip Paeps • Secretary of the FreeBSD Foundation Board, FreeBSD Committer, and Independent Consultant.

Kristof Provost • Treasurer of the EuroBSDCon Foundation, FreeBSD Committer, and Independent Consultant.

Hiroki Sato • Director of the FreeBSD Foundation Board, Chair of Asia BSDCon, member of the FreeBSD Core Team, and Assistant Professor at Tokyo Institute of Technology.

Benedict Reuschling • Vice President of the FreeBSD Foundation Board, a FreeBSD Documentation Committer, and member of the FreeBSD Core Team.

Robert N. M. Watson • Director of the FreeBSD Foundation Board, Founder of the TrustedBSD Project, and University Senior Lecturer at the University of Cambridge.

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Copy Editor • Annaliese Jakimides

Design & Production • Reuter & Associates

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org
Copyright © 2020 by FreeBSD Foundation. All rights reserved. This magazine may not be reproduced in whole or in part without written permission from the publisher.

LETTER from the Foundation

More Online Events Coming Soon

Welcome to the May/June 2020 issue of the *FreeBSD Journal*. It's hard to believe we're nearly halfway through 2020. Since you already know that we're facing unprecedented times, we just wanted to take a moment to wish you well as we enter the second half of the year.

For those of you missing the in-person learning events, members of the FreeBSD Community and FreeBSD Foundation have worked together to come up with a few alternatives. At the end of April, Allan Jude launched the FreeBSD Office Hours series. These 1-hour sessions take place every other Wednesday at 18:00 UTC, and may consist of general Q&A, demos, or tutorials. Videos of past Office Hours can be found on the FreeBSD Project YouTube channel. Upcoming events can be found here. If you would like to run an office hour, please check out the wiki to sign up.

Beginning July 17, the Foundation is launching FreeBSD Fridays. It's a series of 1-hour live 101-type classes. The series begins with Deb Goodkin's *Introduction to FreeBSD*. More information can be found here.

Finally, the *Journal* team continues to work hard to provide excellent educational FreeBSD content. We hope you enjoy and share this issue.

On behalf of the FreeBSD Foundation,
Anne Dickison

FreeBSD® JOURNAL
May/June 2020

12 **TLS Offload in the Kernel**
TLS offload permits the kernel to send and receive data over a socket using TLS where the TLS framing and encryption is performed in the kernel rather than userland. *By John Baldwin*

24 **Interview: Michael W Lucas**
Excerpts from a BSDNow conversation with Michael W Lucas about the second edition of his book, *Sudo Mastery*, and his list of potential future projects. *By Allan Jude and Benedict Reuschling*

31 **FreeBSD Desktop Distributions**
From integrated desktop environments and portable USB-based systems to data recovery tools, FreeBSD has been used by a wide variety of open-source projects. *By Drew Gurkowski*

35 **Bulk Ports Management with Poudriere**
Poudriere is an extremely powerful tool designed for package production, but it can also be used to bulk manage ports. *By Drew Gurkowski*

3 **Foundation Letter**
More Online Events Coming Soon *By Anne Dickison*

5 **New Faces of FreeBSD**
In this installment, the spotlight is on Nick O'Brien, who received his src bit in March, and Richard Scheffenegger, who received his src bit in April. *By Dru Lavigne*

9 **We Get Letters**
Dear Sudden Telecommuter *By Michael W Lucas*

39 **Events Calendar**
We're building on our FreeBSD Fridays content to offer 1-hour live sessions every Friday that are designed to get you started with FreeBSD. *By Anne Dickison*

new faces of FreeBSD

BY DRU LAVIGNE

This column aims to shine a spotlight on contributors who recently received their commit bit and to introduce them to the FreeBSD community.

In this installment, the spotlight is on Nick O'Brien, who received his src bit in March, and Richard Scheffenegger, who received his src bit in April.

Tell us a bit about yourself, your background, and your interests.

• Nick: I was born in and grew up in Seattle, Washington, but I've been living in San Jose, California, for the most part since late 2017. Currently, I work on embedded software at Axiado where we get to use FreeBSD to do RISC-V things.

Initially, I got into writing software more as a necessity than as a direct interest in writing code. Since the time I was in grade school, I have needed to write software for different personal projects and would just Google "How to ...." After years of miscellaneous software development, I began to realize that I really enjoyed the actual problem-solving aspect of engineering and got increasingly serious about the actual skill of writing software.

Some of my hobbies outside of computers include running, getting thrashed by waves when trying to surf, and strumming a guitar.

• Richard: I'm based in Vienna, Austria (commonly mangled up with Vienna, Virginia, here at my workplace), which is also my hometown. Some 15 years ago, I joined a Network Storage Company that was looking for networking experts to support their then new product, a caching web proxy. However, despite my skills in that area, my resume was turned down by a professional headhunter. I started my networking career by setting up a "networked" BBS (part of the TrekNet, based on FIDOnet technology) while in school, using mail and USENIX gateways before the general population even knew the Internet existed. This was well before the web became a thing. I remember that Gopher was all the rage when I visited a computer room at the university. Then I worked in a number of smaller companies, always related somehow to networking. At that time, around the turn of the millennium, I was the go-to expert in German-speaking countries for one vendor's special network operating mode, a way-too-early precursor to TRILL—where even the vendor's support people reached out to me numerous times. But before digressing too much, by chance, my resume was picked up by an intern working at the headhunter company and simply filed my profile for a similar job, in addition to the job I originally applied for.

After having been thoroughly tested and interviewed multiple times (in retrospect, I believe the people doing the technical part could not figure out how I responded so knowledgeably and quickly, while none of my answers could be found verbatim by any of the search engines

of the day), I started at Network Appliance, now known as NetApp, in 2005. My original office back then was the EMEA headquarters in Amsterdam, where I worked in a field related to my prior IT expertise. NetApp is one of a handful of enterprise-class vendors offering professional, large-scale NAS solutions. Most other large storage vendors emphasize SAN much more, so my ending up at NetApp was a good match after all. Eventually, I started working in an EMEA-wide role within the company and was allowed to pick my primary office. So, after having worked out of Germany, I eventually moved back to where I came from.

Also, being very interested in networking—in particular, transporting data from A to B—turns out to be quite an important expertise when your employer deals with devices that are supposed to provide data as quickly as possible across a variety of different protocols—NAS and SAN, and for the IP part of that, via NFS, SMB, and iSCSI (NVMe coming soon to IP too). And, as I had been trained as a chemist, a good grasp of complex systems interactions also helped.

Over the course of my tenure, I became almost obsessed with trying to eradicate all sources of excessive delay in IP protocols, which makes people often look for different technological solutions when it comes to storage (e.g., SAN, IB). I remember getting our CTO at that time upset, as I wanted to convince him in a prepared speech in an internal forum with a larger audience about the merits of reduced IP latency for our products—after having had a short chat in private with him, in which he agreed with me.

The interaction led me to engage with the IETF as an active participant, and I am still working with a heavy focus in the transport area and TCP, in particular. For a short while, I was also cochair of the (newer) AQM Working Group during its existence and worked to get more modern Queue management mechanisms than RED adopted. The reason for my involvement in AQM is that I had become quite interested in Explicit Congestion Notification, which is a way for TCP to adjust without loss to the (momentary) capabilities and capacities of the network. And it is generally understood that what most network devices implement—if they implement something other than drop-tail—is rather inadequate for the intended task.

However, as we're located in Europe quite far from our engineering, which is in the U.S. and India, the pace of improvements in our product in the space I cared about was slow. At some point, when I provided yet another fully worked-out bug report around TCP to address some issues I had observed with our customers running storage traffic across TCP/IP, I was told "Once there's a FreeBSD patch available for this behavior, we can pull it in."

So that is how I really got involved with the FreeBSD Project, by reaching out to my existing contacts and working to become first a contributor (providing patches with test cases, explanations about where the issues are, and also filing some bug reports) and then ultimately being accepted as a committer.

How did you first learn about FreeBSD and what about FreeBSD interested you?

• Nick: I had heard about FreeBSD here and there for a while and probably even used it, but I didn't start really getting into FreeBSD until I joined Axiado in 2018, where we started with figuring out how to get FreeBSD booting on RISC-V hardware. I enjoy writing code for FreeBSD, but I also really like the dedicated, tight-knit community. It feels like it's a relatively small community that's making quite a large impact. Lastly, I couldn't pass up an opportunity to work with so many smart people.

• Richard: It is hard to say; I was certainly aware of the BSD family for at least the last 20 years. My first active exposure to FreeBSD was with what became https://reviews.freebsd.org/rS206456. I became interested in TCP ECN and found that the original ECN implementation in the BSD family suffered from an oversight (and not fully documented behavior in RFC3168) where it would not behave as expected. As FreeBSD is deeply ingrained in some of the commercial products of my employer, most notably the ONTAP data management software, it naturally followed that I grew increasingly interested in improving the situation there. That was particularly due to the fact that FreeBSD used to lack a number of important features (such as DSACK), which are important in interfacing with systems in the cloud, or had features which were not quite on production level (e.g., CUBIC). The latter is quite important for delivering good performance across longer distances. NetApp offers a solution built around synchronous data replication across wide distances called MetroCluster, which is my product expertise. Even while CUBIC has been available for some time, storage class traffic has its own set of challenges. For one, storage IO is very transactional—a client issues very detailed requests. Also, storage IO is often application limited—data is only requested at the pace the application can process it and a server can't transmit a large object as quickly as possible as it is only determined by TCP and network limits. This is quite unlike the typical architectural use case of TCP (infinite data and as fast as the network/receiver allow) and also is not encountered in a more web-dominated environment.

How did you end up becoming a committer?

• Nick: Kristof Provost (kp@) and Philip Paeps (philip@) do FreeBSD consulting work at Axiado. After working with them for some time and after I had interacted with the RISC-V community on Phabricator, they invited me to join my first developer conference in San Jose at Intel in October of 2019! It was a great experience, and that was really the first time I had seen a team come together like that for an open-source project. After a few more months on Phabricator, they proposed a commit bit for me. I was ecstatic!

• Richard: I became familiar with a number of committers to the FreeBSD Project during my IETF meetings. Working on improvements on TCP in general also entails implementing and deploying them eventually. So, I was already writing code to improve SACK loss recovery that was based on a suggestion of mine that got fully worked out in RFC6675, known as rescue retransmission. Most of that got stashed away in private builds and is not universally available. Working at a data company also exposes you to interfacing with appliances and software provided by many other vendors, some of which are also making use of FreeBSD, or parts thereof, in their software. I wanted to contribute bug fixes and improvements in the larger ecosystem for other users of FreeBSD—and ultimately also benefiting my employer and the mutual customers of those other vendors—by generally improving the behavior, stability, and interoperability in the still typical, complex, heterogeneous environments that are commercial IT architectures. After providing around 40 patches (mostly bug fixes and one major feature—DSACK—which resonated well with other contributors), I was invited to become a committer with rgrimes@ and tuexen@ as mentors. I'm still in awe at having that significant level of process and technical expertise to help me improve my contributions to the project.

How has your experience been since joining the FreeBSD Project? Do you have any advice for readers who may be interested in also becoming a FreeBSD committer?

• Nick: It has been a really great experience, but it was also a slightly weird time to join with a lot of strange world events taking place. My favorite part so far is bouncing ideas back and forth with different FreeBSD developers all over the world. Everyone seems open to listening, so I feel like I really have a voice. I've also learned so much from my peers!

My advice to readers who would like to become committers is to understand that it's not too far out of reach. You don't need to get struck by lightning to get selected as a committer. Just start out by deciding what you'd like to contribute to, follow the ongoing upstream work on Phabricator, connect with others already working on those bits, and chip in where you can! When I told him I was interested in becoming a committer, I remember philip@ said something like, "It's not as far off as you think." At the time, it really didn't even feel like it was a possibility, but he was right; just a few months later I was a committer! We're all a team and everyone is here to help so just be grateful for those supporting you and try to make the most of it! If you're really trying to become a committer, trust me, someone will notice.

• Richard: Well, the core transport@ group (dealing mostly with TCP, some SCTP, and related aspects) holds conference calls every other week. I started participating in them when I began contributing proper diffs on reviews.freebsd.org. These conference calls are very helpful in discussing problems and solutions, as well as building a reputation. As you can imagine, messing around with something as critical as TCP comes with a very high bar to not introduce any regressions or performance degradations. Having said that, one of my earlier commits did have some unintended side effects, so I learned in a flash how to revert commits from HEAD. The review process there is very strict, and patches get discussed at length before being accepted. In short, I would not recommend anyone to pick the base TCP stack as their focus area if their interest is more about learning the processes of working in an open-source project. However, the peers and mentors I found are all excellent and give very good feedback. Counting my earliest, indirect contribution to the project with the ECN issue, it was about 10 years until I became a contributor. In my specific case, my employer has been supportive, as enhancing the status quo of transport in the networking space is important from a product perspective—even when it's a very slow process to make sure nothing breaks along the way. But the length of time it took me to become a committer is atypical and more about the area of my contributions. I know a few committers who started indirectly contributing to the FreeBSD Project while working on their master theses—when their professors were supportive—and they still contribute to this day. Since a master thesis hardly ever takes more than a year to complete, that would be more the norm.

DRU LAVIGNE is the author of *BSD Hacks* and *The Best of FreeBSD Basics*.

WeGet letters

by Michael W Lucas

Oh, Transcendent All-Knowing Letters Person,

We're all working from home now, and the network is terrible. It doesn't matter if I'm using my cable modem or my cellular hotspot, working on the office network is slow and painful and super annoying. Even the Internet seems clunky. What can I do?

—Sudden Telecommuter

ST,

You can accept that you have been consigned to an inner ring of the Inferno and get on with shambling through your wretched existence.

No, no, fine. I know how this goes. I give you a shot glass brimming—brimming, I say!—with undiluted truth, and you write me back beseeching me to break it up into sugar-coated capsules that'll go down without you having to taste them. It's as if you want to guzzle bridge mix without tasting any of the delicious licorice, when the licorice is the ingredient that builds character and integrity. No, not "black licorice." That's like saying "wet water." There is only one licorice. We do not speak of the loathsome, extruded-sugar rainbows that have reprehensibly stolen licorice's wholesome banner for their own unspeakable enterprises.

So. Let's see if I can break this down into drams sufficiently minuscule for your tender palate.

The Internet isn't a single network. It's a collection of interconnected networks. The miracle of this isn't that the networks can interoperate. No, once you consider that the overwhelming majority of these networks are run by for-profit companies intent on providing service at the lowest possible price point, the miracle is that these networks function even internally. Kludges and workarounds suffuse them all. A network that isn't running at full capacity wastes resources, namely the network operator's money. They're designed to have corporate users during the day and residential users on evenings and weekends—but with the unexpected immediacy of an asteroid impact in Lake Michigan, people's homes have started gobbling bandwidth during the day. It's no different than the shortage of home toilet paper triggered by millions of employees unexpectedly being denied the fringe benefit of their sumptuous corporate facilities.

And you want to know what you can do about it?

My suggestions might appear intended to help you maintain that brittle façade of empowerment. Don't let them fool you. They're all about methodically and inexorably demolishing each board in that front, so that you can fully embrace the existential horror of the modern Internet.

First, understand that your ISP has sold you the sort of lie that would make the most gloriously demonic used car salesperson blush from shame. Perhaps the link into your house is full-on gigabit. Perhaps it's not the sort of gigabit that's a technicality because it speaks the gigabit protocol, but it can actually push almost a thousand megabits a second even when your appli-

cation carves up the traffic into puny little 64-byte chunks and spews millions of them. Sure, you can get those to the ISP's router. But the ISP sold every house on your block one of those gigabit lines, and their connection to their upstream is, what, 10 gigabit? How many houses are in your neighborhood? The ISP's network is optimized for seamless delivery of The Wiggles, because the list of Customers You Don't Want to Hear From starts with "overwhelmed single parent desperately seeking socially-acceptable spawn sedation."

To see exactly how doomed you are, break out MTR. The acronym might have originally stood for "Matt's Trace Route," but it's been repeatedly retronymed and "Multiple Trace Route" seems to be the interpretation of the day. MTR gives you a real-time view of how traffic flows between network nodes. A node that appears as a series of asterisks, or with very high times, isn't necessarily bad; it simply means that the node prioritizes supporting actual traffic over your petty, irrelevant, and unwanted diagnostics. MTR runs constantly, so you can keep it going in another window. When a network connection hangs up on you, you can flip over to your MTR session to see who's dropping packets.

If you're so desperate as to think reporting a problem will give you relief, MTR output is essential. Your ISP's helpdesk will be happy to educate you why you're wrong.

Mind you, that output's largely useless without a corresponding MTR going the other way. Running MTR on your remote node reveals how traffic goes from that node back to you. It might follow the same path. It might traverse a completely different set of routers. If an ISP reboots a router, it might all flicker and fade and founder, or it might seamlessly switch. Who knows? Look both ways before crossing the Internet.

If MTR doesn't expose any ghastly secrets, you aren't looking hard enough. But let's try another tool.

Many network problems occur at your home. The neighbor's need for a microwaved burrito might make your wireless cut out. Squirrels nesting in the cable box on the phone pole find data frames highly tasty. Fortunately, you have the tools to see where such problems occur. Most computing professionals have a bunch of technology in their homes. You're probably streaming music from the Internet or your own server, or have a movie on in the background, or something. Use these to narrow the problem scope.

If you have traffic, you can scrutinize it. All computing professionals must have a passing familiarity with packet sniffers. Even a base familiarity with tcpdump(8) will serve you well. If you want a glossier tool, try Wireshark. A book like Sanders' *Practical Packet Analysis* will help you achieve base competence.

Speaking of packet sniffers, consider what your computer is putting on the network. Even Unix desktops can unpleasantly surprise you with the garbage they spew, and if you're running a commercial operating system you're almost certain to see it spewing useless traffic. (Useless to you, that is. The software developer finds it highly monetizable.) Also consider what traffic arrives at your host. Do you have an IoT device constantly vomiting multicast that nobody asked for? Is your buggy switch sending megabits of bootp requests a second? I've seen these and worse. Individually these are small, but they all demand that every device on your network expend precious computing resources rejecting and ignoring it all. Clean up your own sewer before you criticize the wastewater treatment plant.

If you don't see an identifiable problem, you might fall back on blind luck. If your IPv4 connection is painfully slow, try using IPv6. If you or your organization don't have IPv6, I invite you to join the 21st century at your earliest convenience. If the IPSec VPN is slow, try SSH or openvpn. Perhaps one of those can leap across the chasms in the lackadaisical collection of kludges we call the Internet.

Ultimately, the less traffic you can send across the network, the happier you will be.

That means SSH.

The command line is everything. Bind yourself to it. Our predecessors thrived on 300 baud. Cease complaining that the network no longer supports you in the manner to which you have grown accustomed and acclimate yourself to your sadly fallen circumstances.

Once you swallow that, you'll be fine.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS (https://mwl.io)'s newest books are *Sudo Mastery, 2nd Edition* and *Terrapin Sky Tango*.

FreeBSD®

The FreeBSD Project is looking for

- Programmers
- Testers
- Researchers
- Tech writers
- Anyone who wants to get involved

Find out more by

Checking out our website
freebsd.org/projects/newbies.html

Downloading the Software
freebsd.org/where.html

We're a welcoming community looking for people like you to help continue developing this robust operating system. Join us!

Already involved?

Don't forget to check out the latest grant opportunities at
freebsdfoundation.org

Help Create the Future. Join the FreeBSD Project!

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system.

Not only is FreeBSD easy to install, but it runs a huge number of applications, offers powerful solutions, and cutting edge features. The best part? It's FREE of charge and comes with full source code.

Did you know that working with a mature, open source project is an excellent way to gain new skills, network with other professionals, and differentiate yourself in a competitive job market? Don't miss this opportunity to work with a diverse and committed community bringing about a better world powered by FreeBSD.

The FreeBSD Community is proudly supported by

FreeBSD FOUNDATION

TLS Offload in the Kernel

BY JOHN BALDWIN

FreeBSD 13.0 adds support for Transport Layer Security (TLS) socket kernel offload. TLS offload permits the kernel to send and receive data over a socket using TLS where the TLS framing and encryption is performed in the kernel rather than userland. Kernel TLS (KTLS) requires changes to userland SSL libraries, the kernel's network stack, and, in some cases, device drivers.

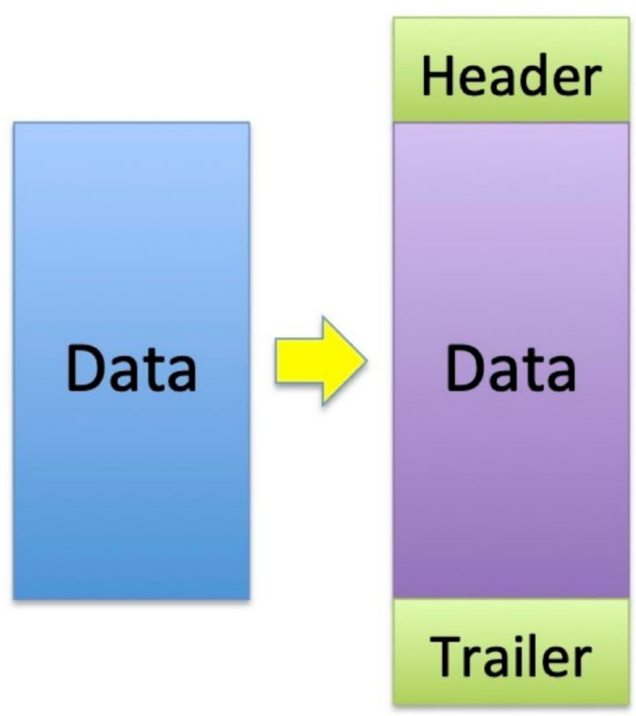
Transport Layer Security

TLS is an application layer protocol designed to provide authentication and privacy to upper layer application protocols. TLS is structured as a stream of records, or frames, transmitted and received over a stream protocol. (There is a version of TLS for datagram protocols, but KTLS only supports TLS over TCP.) Each TLS record contains a header with a message type and length. This record layer is used to transport messages defined by the TLS protocol to manage the TLS connection as well as application data messages that carry data from an upper layer application protocol such as HTTP, SMTP, or IMAP.

A typical TLS connection begins with a few TLS protocol messages used to establish a TLS session. These messages are used to verify the identity of the other end of the connection, choose a set of encryption and authentication algorithms (known as a **cipher suite**), and establish a pair of ephemeral session keys used to encrypt and authenticate subsequent TLS records. Once the connection is established, the connection switches to passing the upper layer application data via application data messages. These application data messages constitute the bulk of the messages in a TLS connection.

Application data messages are built from variable-sized blocks of application data. The application data is encrypted and headers and trailers are added to form a TLS record (see Figure 1). TLS protocol messages are constructed similarly except that the TLS header contains a different message type and the payload data of the TLS record comes from the TLS protocol. The format of the TLS header and trailer are determined by the specific cipher suite used by the session.

Figure 1: Constructing a TLS Record



Why Move TLS into the Kernel?

Normally we avoid putting code into the kernel when possible. User code runs with less privilege and is isolated from other processes. A crash in a user process only affects the process, and vulnerabilities are constrained to a specific process. Kernel code runs with more privilege and is able to access all data in the system. Bugs in kernel code have more significant consequences. Given that, why add the additional complexity of TLS to the kernel? The reason, as with most other code that resides in the kernel, is performance.

There are two main justifications for KTLS. The first is to avoid extra copies of data in and out of the kernel. The second is to enable the use of TLS offload in network interface cards.

Avoiding Data Copies

The initial work on KTLS was motivated by regaining the zero-copy performance of `sendfile()` when using HTTP over TLS. Prior to the advent of `sendfile()`, the typical workflow of FTP and HTTP servers required reading the contents of files into buffers in a user process via `read()` followed by a call to `write()` to send the data out through a socket (see Figure 2). This resulted in three copies of the data: one in the kernel's buffer cache (or VM page cache) associated with the file, a second in temporary buffers in userland, and a third in the socket buffer. The `sendfile()` system call instructs the kernel to send a portion of a file's contents over a socket. With this higher-level request, the kernel is able to reuse the copy of the file data in the buffer cache directly in the socket's send buffer eliminating the additional copies of the file data (see Figure 3).

Figure 2: Pre-sendfile() HTTP/FTP Workflow

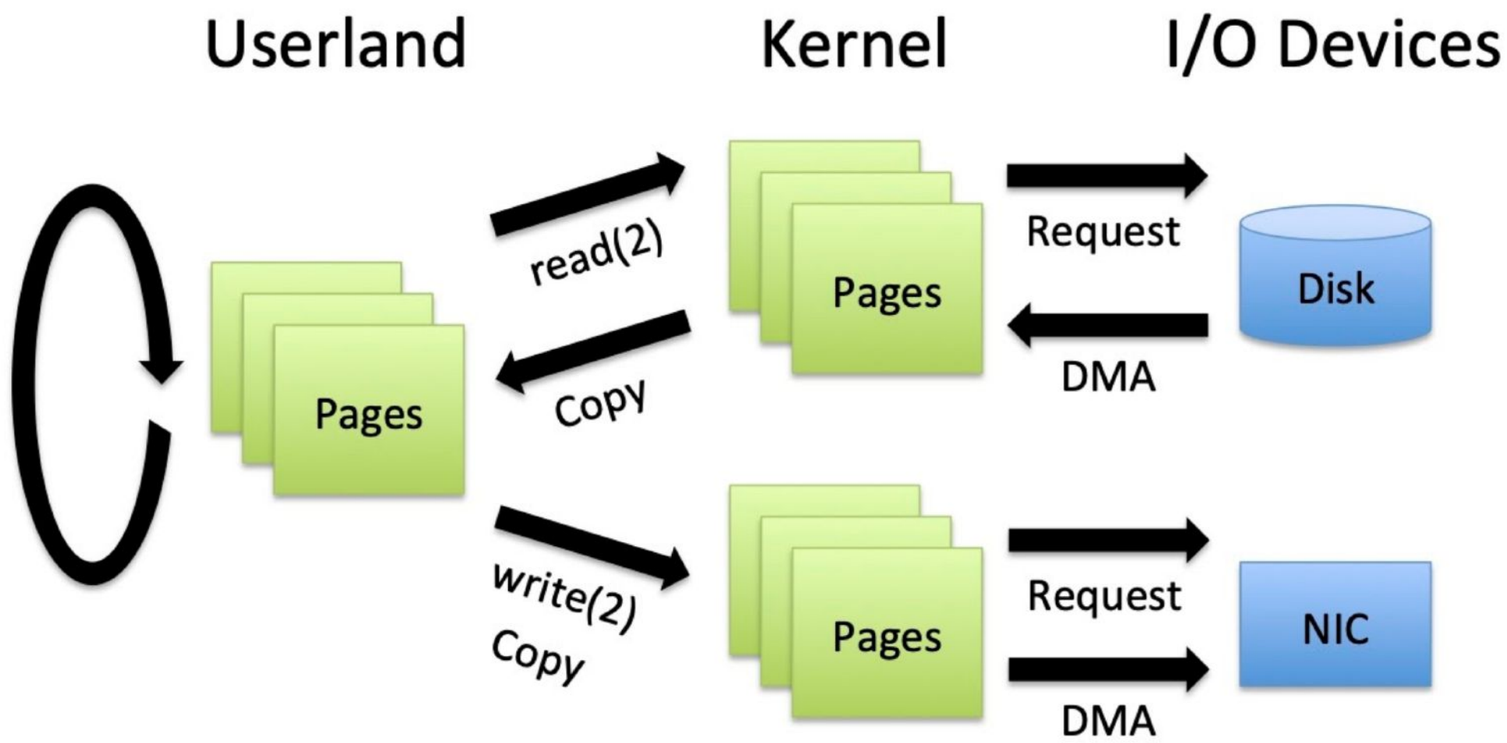
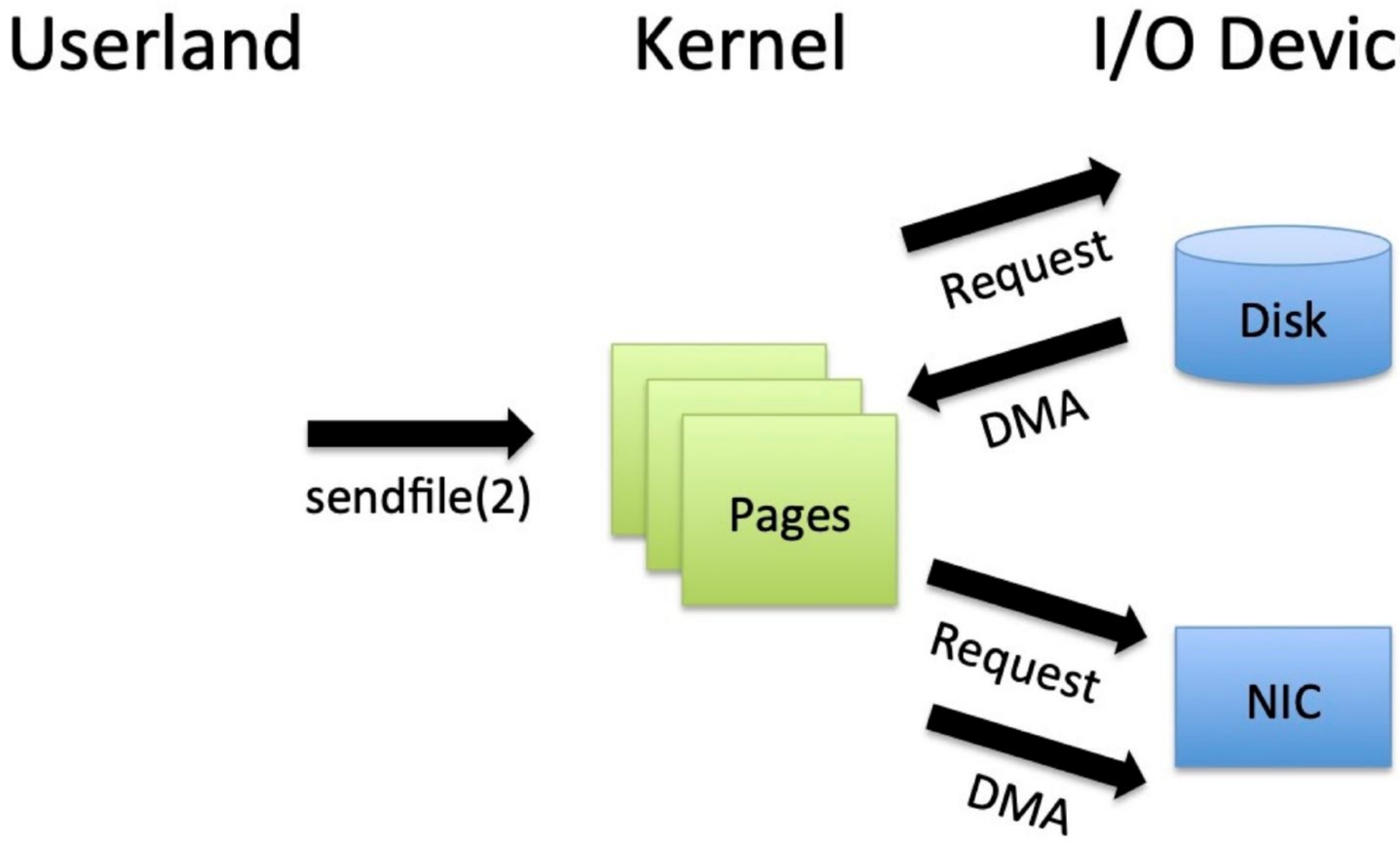


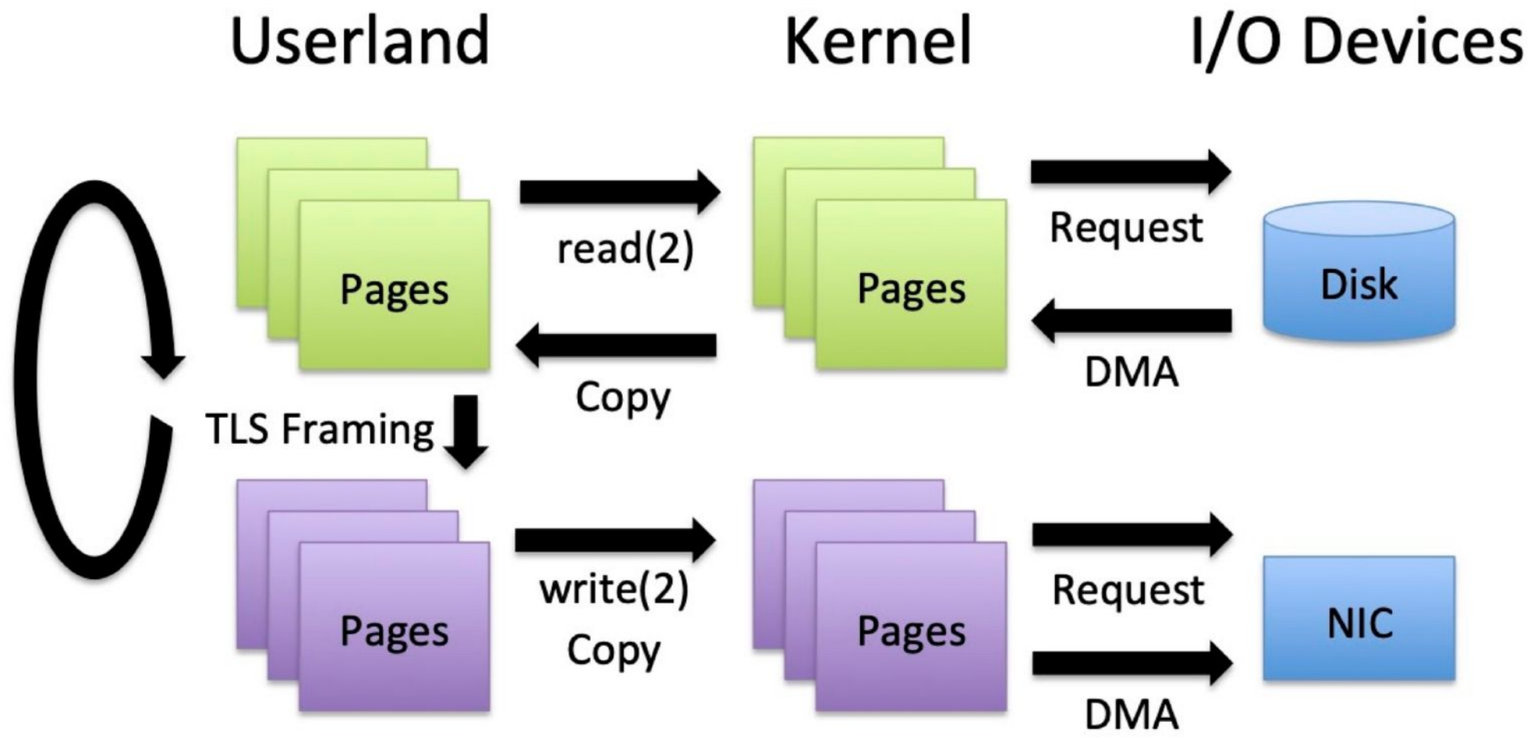
Figure 3: sendfile() HTTP/FTP Workflow



With HTTP over TLS, the `sendfile()` system call can no longer be used as-is. The data sent over the socket is not an exact copy of the file's contents. Instead, the file data must be encrypted and encapsulated in TLS frames. With TLS framing performed in userland, this requires a return to the pre-`sendfile()` workflow. Data is copied from the buffer cache into temporary buffers in userland where it is encrypted and framed. This modified data is then copied into the

kernel's socket buffer (see Figure 4). By moving TLS awareness into the kernel, we can avoid one or both of these extra copies.

Figure 4: HTTPS Workflow with User TLS



A similar issue exists with a more recent workflow: NFS over TLS. The NFS client and server run in the kernel. They directly move data for files between the buffer cache and socket buffers. While the client and server do copy data between the buffer cache and socket buffers, performing TLS in userland would require additional copies into and out of userland.

TLS Offload in NICs

A couple of recent Ethernet cards including the Chelsio T6 and Mellanox ConnectX-6 Dx include support for inline encryption and decryption of TLS records. For transmit, this permits the host to send unencrypted data to the NIC, and the NIC will encrypt the data and split it into multiple TCP segments before transmitting the data on the wire. Similarly, for the receive case the NIC assembles multiple TCP segments into a TLS record and supplies the decrypted TLS record to the host. The driver is responsible for providing the session keys to the NIC to permit the inline encryption and decryption.

KTLS Requirements

These use cases have similar requirements. `sendfile()` requires the kernel to have control over TLS framing including sequence numbers. Since the sequence number is an input into the authentication algorithms used in TLS, this in turn requires the kernel to manage the encryption of all TLS frames on the socket for `sendfile()`. Encrypting all TLS frames requires userland to provide unencrypted data for all TLS records to the kernel. NFS over TLS similarly requires full control over TLS framing in the kernel. TLS offload in NICs requires access to unencrypted data for all TLS frames on the socket.

In short, these use cases require moving the TLS record layer into the kernel. However, none of these use cases require moving the handling of TLS protocol messages, such as those used to establish a TLS session and negotiate session keys, into the kernel. As a result, KTLS moves the handling of the TLS record layer into the kernel but leaves the management of TLS protocol messages in existing userland SSL libraries.

This is similar to the approach used with IPsec where key negotiation is performed by user-land daemons, but individual packets are encrypted and decrypted in the kernel. For IPsec, the keys are negotiated over a separate, side-band connection and are shared across several connections. For TLS, the keys are unique to a connection and are negotiated within the connection via TLS protocol messages. In addition, TLS records and IPsec packets are encrypted and authenticated by many of the same algorithms.

KTLS Implementation

KTLS in the kernel consists of three main components. TLS sessions store information about one direction of a TLS connection including the cipher suite used, session keys, and a backend used to encrypt or decrypt TLS records. KTLS also hooks into the network stack transmit and receive paths. While the transmit and receive paths do share some properties, they are implemented differently.

TLS Sessions

TLS sessions manage the encryption and decryption of TLS records. The send and receive sides of a TLS connection are managed independently with separate TLS sessions for transmit and receive. Each session is described by a `struct tls_session` object and contains information about the session such as the version of TLS, the cipher suite, and the encryption and authentication keys. Sessions are also associated with a backend which performs the encryption and decryption.

Three different types of session backends are supported. Software backends encrypt and decrypt TLS records while they are present in the socket buffer. They require no knowledge of TLS outside of the socket layer. Specifically, protocols such as TCP and device drivers do not require any changes to support software backends. NIC TLS backends encrypt and decrypt TLS records in a network card as part of transmitting or receiving packets. This requires cooperation with the protocol layers as well as explicit support in device drivers. Finally, TOE TLS backends work similarly to NIC TLS except that they leverage TOE support to manage TCP state management such as retransmits.

KTLS generally treats the send and receive sides of a TLS connection independently. An individual connection may only offload one side instead of both. In addition, an individual connection may use different types of backends for each side. For example, connections may offload TLS transmit in the kernel while handling TLS receive in the userland SSL library, or a connection may use NIC TLS to offload TLS transmit and a software backend to handle TLS receive.

TLS Transmit

The initial work on KTLS focused on offloading encryption of transmitted TLS records. HTTPS server workloads of static content generally transmit significantly more data than they receive. As a result, the biggest performance gains for these workloads come from offloading encryption of transmitted data rather than decryption of received data.

When TLS transmit is offloaded to the kernel, userland applications always provide unencrypted data to the kernel. Data sent via system calls such as `write()` or `sendfile()` are split into separate TLS records by the kernel. These TLS records always use the application data message type. Userland can use the `sendmsg()` system call with a `TLS_SET_RECORD_TYPE` control message to send individual TLS records with a custom type and size. For these requests, the contents of the data described by scatter/gather list in the message header is sent in a single

TLS record with the message type given in the control message. This is used by the SSL library to send TLS protocol messages such as handshake messages and alerts.

A TLS transmit session is created by a `TCP_TXTLS_ENABLE` socket option. A userland SSL library invokes `setsockopt()` with this option supplying an instance of `struct tls_enable` as the option value. This structure includes pointers to the session keys and a description of the negotiated cipher suite and TLS protocol version. The socket option handler in the kernel creates a new TLS session object and probes for a backend. `TOE TLS` and `NIC TLS` backends are probed by calling down to the device driver for the NIC associated with this connection. If the NIC driver does not support TLS or is not able to offload the connection, the handler searches for a software backend. If no backend is found, the `setsockopt()` system call fails and the SSL library continues to perform TLS encryption in userland. If a backend is found, a reference to the new TLS session is saved in the socket's send buffer. Existing data in the socket send buffer is transmitted as-is, but all subsequently written data to the socket buffer is encapsulated in TLS records and encrypted.

TLS mbufs

Each transmitted TLS record is described by a single `struct mbuf`. TLS mbufs use a new **external pages** mbuf added in FreeBSD 13. External pages mbufs do not store payload data in a virtually contiguous buffer pointed to by `m_data`. Instead, these mbufs contain an array of physical address pointers to one or more pages in RAM. This does mean that the traditional way to access mbuf data, the `mtod()` macro, cannot be used with these mbufs. Kernel code paths using these mbufs were extensively audited for code using `mtod()`.

These mbufs were first added to improve the performance of `sendfile()`. Previously, `sendfile()` used a separate mbuf for each page in a file. With external pages mbufs, a single mbuf can describe multiple pages. This permits a smaller number of mbufs to describe the same amount of file data in a socket buffer. Fewer mbufs means fewer cache misses when walking the linked list of mbufs in the socket buffer which improves performance.

Network interface device drivers can choose to support transmitting these types of mbufs by advertising support for the NOMAP capability. For device drivers which use existing `bus_dma` routines to map mbufs without examining the contents of mbufs being transmitted, no further changes are needed apart from setting `IFCAP_NOMAP` in both `if_capabilities` and `if_capenable` during their attach routines. Note that a device driver must be able to offload checksums for packets containing these mbufs, but checksum offloading is a widely supported feature in NICs. If a device driver does not support external pages mbufs or it does not support necessary checksum offloading, then the network stack will convert external pages mbufs into a chain of conventional mbufs before passing them to the device driver.

TLS mbufs extend external pages mbufs to store TLS-specific information such as the TLS record header and trailer and a reference to a TLS session. Device drivers which support external pages mbufs must also support transmitting the data stored in the TLS record header and trailer fields. However, for most device drivers this does not require any specific changes since existing bus_dma routines already handle these fields.

The `ktls_frame()` function sets TLS-specific information for TLS mbufs. This function uses the TLS session associated with the socket's send buffer to construct the record. It stores a reference to this TLS session in the mbuf and populates the TLS header of the record including any explicit IV or nonce. Finally, this function computes the length of the TLS trailer accounting for any required padding. The TLS trailer's contents are not set until the TLS record is encrypted but setting the trailer length ensures that the mbuf reserves the correct amount of space in TCP sequence numbers.

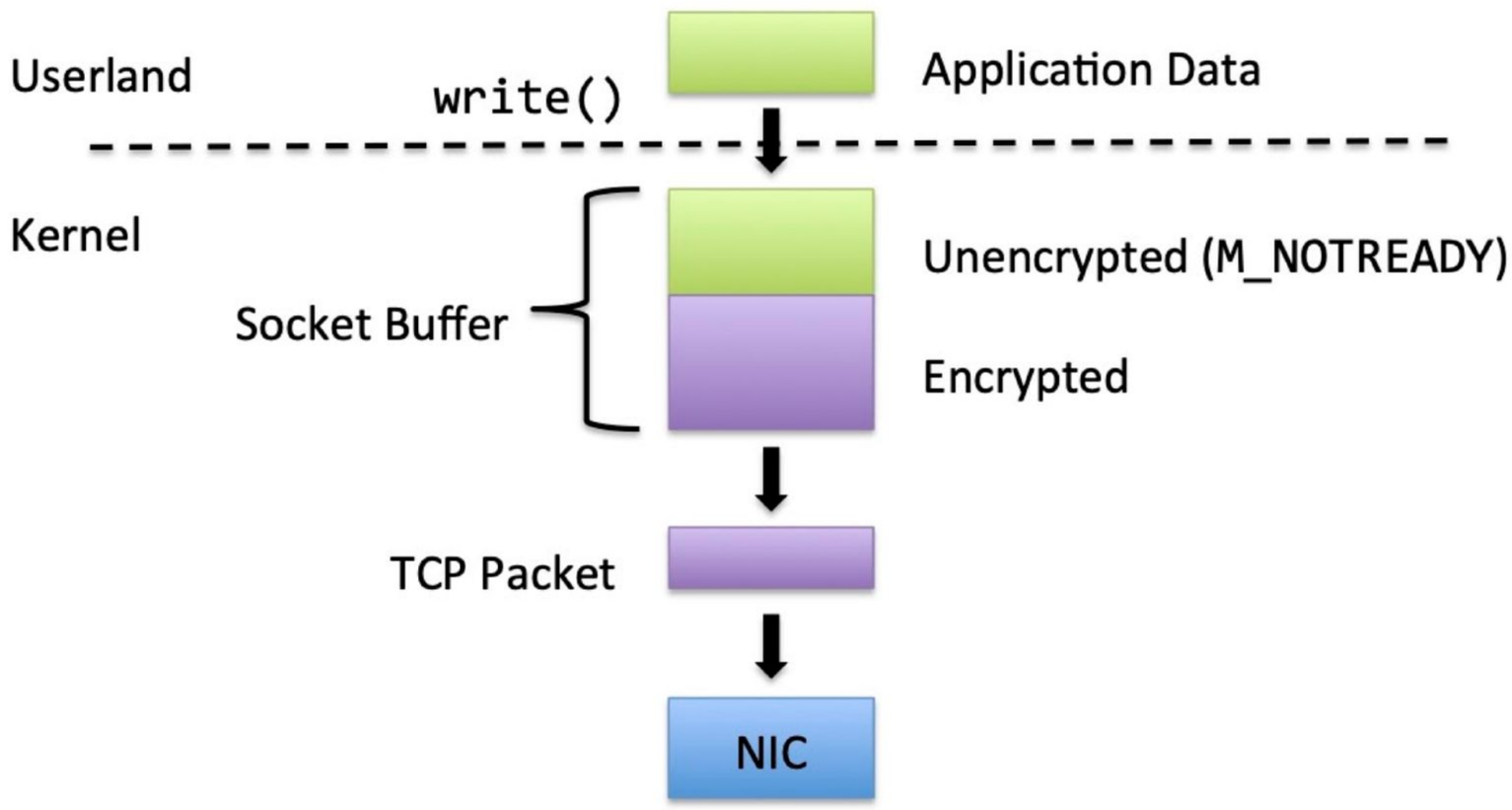
Software TLS Transmit

When software backends are used, TLS transmit encrypts TLS records in the socket buffer before they are released to the transport protocol. In the socket buffer this relies on the existing `M_NOTREADY` mbuf flag.

The `M_NOTREADY` mbuf flag is used to mark mbufs in a socket buffer that do not yet contain valid data. These mbufs still reserve space in the socket buffer to provide backpressure to whatever is producing data, but they cannot be used by the socket buffer consumer. The `sendfile()` system call uses these mbufs to reserve space in a send socket buffer for the regions of a file that are not already in memory at the time of the system call. Instead, disk I/O requests are scheduled to populate these missing pages. When the I/O requests complete, the mbufs are marked ready by clearing the `M_NOTREADY` flag, and the protocol is notified that new data is available to send from the socket buffer.

Software TLS transmit reuses this same framework to handle TLS records to differentiate unencrypted TLS records from encrypted TLS records. When an unencrypted TLS record is queued to the socket buffer by a system call such as `write()` or `sendfile()`, the `M_NOTREADY` flag is set on the mbuf describing the TLS record by `ktls_frame()`. In addition, the mbuf is placed on a queue of unencrypted TLS records. A worker pool of threads (one per CPU) services this queue. The worker threads invoke the software backend to encrypt each TLS record. Once the TLS record has been encrypted, the mbuf associated with the TLS record is marked ready (see Figure 5). At this point, the TLS mbuf is now a "regular" external pages mbuf. None of the network stack from the protocol layer down to the device driver has to perform any additional work to support TLS when using a software backend. The caveat about external pages mbufs being converted to chains of conventional mbufs from above still apply, but that can be trivially mediated in most device drivers by supporting external pages mbufs.

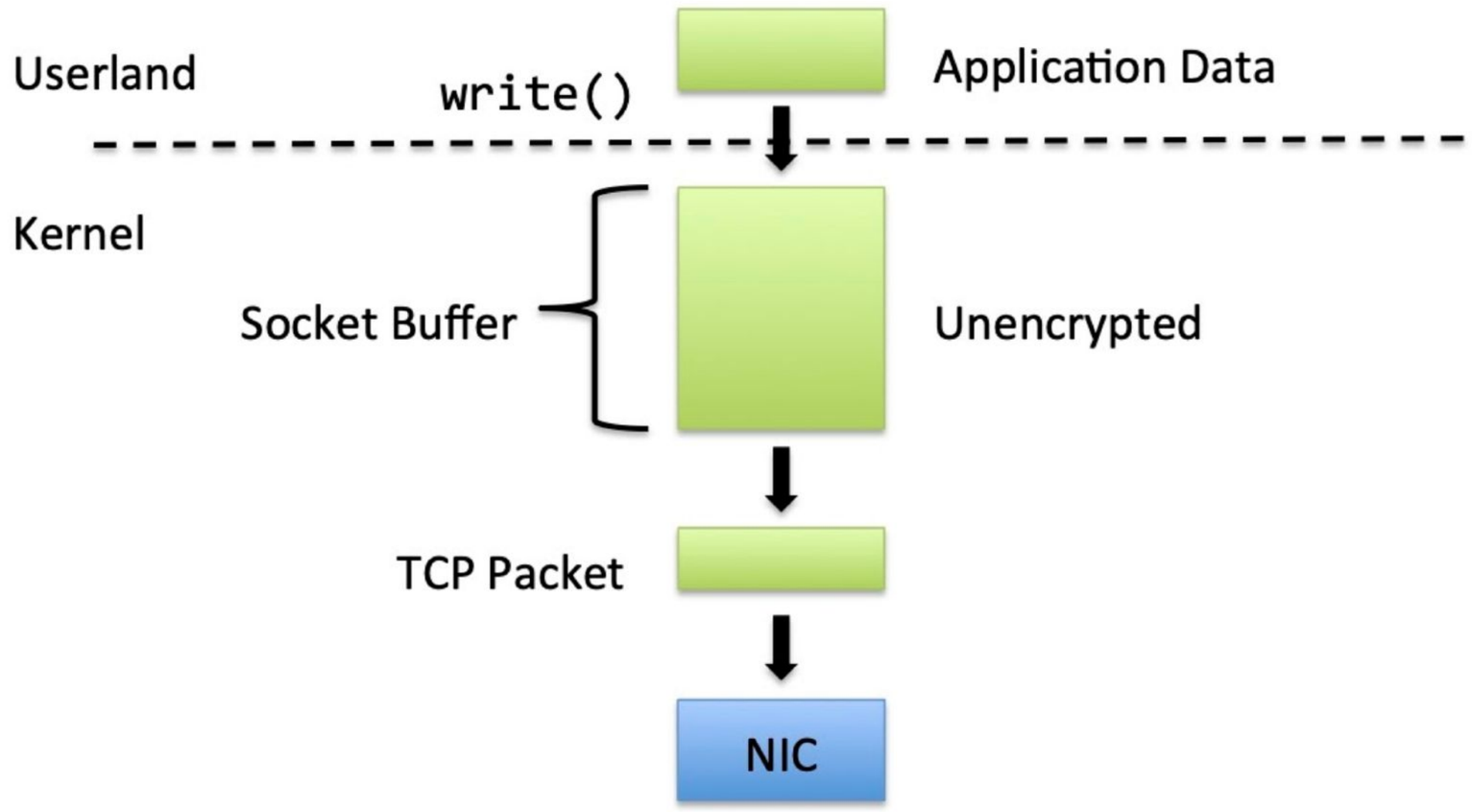
Figure 5: Software TLS Transmit Overview



NIC TLS Transmit

NIC TLS backends require additional changes in the network stack and device drivers beyond the changes needed for software TLS. With NIC TLS, unencrypted TLS records are passed down through the network stack to the device driver. The TLS record payload is encrypted by the NIC after it has been read from host memory via DMA but before it is transmitted on the wire (see Figure 6).

Figure 6: NIC TLS Transmit Overview



To probe for a NIC TLS backend, the handler for the `TCP_TXTLS_ENABLE` socket option uses the route associated with the socket to lookup the network interface servicing the socket. The handler attempts to allocate an `IF_SND_TAG_TLS` send tag from the interface. This send tag will be set on the first mbuf in an mbuf chain that contains one or more TLS records from this session.

To pass unencrypted TLS records down to the NIC, the `ktls_frame()` function does not set the `M_NOTREADY` flag on TLS mbufs when using a NIC TLS backend. NIC TLS mbufs are passed directly to the transport protocol at the time they are queued to the socket buffer.

NIC TLS does require a few changes in the TCP and IP protocols. First, TCP must ensure that it does not mix "plain" data with TLS records in a single packet. In general, TCP is permitted to construct packets which span multiple mbufs in a socket buffer. With KTLS, a connection initially contains several "plain" data mbufs. Once TLS transmit is enabled, all future mbufs in the socket buffer will contain TLS records. This means that there is a brief window after TLS transmit is enabled when the socket buffer contains both types of mbufs. To simplify support for NIC TLS farther down the stack, TCP does not send packets which contain both types of mbufs. This is handled in the `tcp_mcopym()` function. Second, IP and IP6 output must set the send tag on the packet header mbuf at the start of each packet before passing the packet down to the device driver. Since TCP does not mix mbufs of different types, IP output can check the first data mbuf in a packet to see if it has a TLS session. If so, it reads the send tag from the TLS session and sets it on the packet header mbuf. This is implemented in the `ip_output_send()` and `ip6_output_send()` functions.

Finally, NIC TLS requires support in device drivers. Device drivers must handle requests to create TLS send tags. The driver checks that it supports the requested TLS version and cipher suite. If so, it allocates a new send tag holding any device-specific state needed for the TLS session. In addition, the driver must recognize incoming packets which specify a TLS send tag and arrange for them to be segmented and encrypted. This is made more complicated by the fact that TCP may choose to send packets which only transmit a portion of a TLS record. Thus, a driver cannot rely on sending complete TLS records for each request. It may need to send the start of a TLS record, the end of a TLS record, or a region in the middle of a TLS record. In addition, since TLS implicitly supplies support for TCP Segmentation Offload (TSO), a single TCP "packet" may span multiple TLS records.

A key feature of TLS mbufs is that each TLS record is described by a single mbuf. This ensures that a NIC driver can always access the entire contents of a TLS record easily when any portion of a TLS record needs to be retransmitted. For example, a NIC may need a record's entire contents to compute the authentication code stored in the TLS record trailer when transiting the end of a TLS record. If a TLS record were split across multiple mbufs, TCP could potentially free mbufs holding ports of a TLS record that have been acknowledged by the remote end. In addition, NIC TLS drivers would need a reliable way to find other mbufs belonging to a TLS record such as holding extra mbuf references in the driver or examining the socket buffer to locate other mbufs belonging to a TLS record. Using a single mbuf for each TLS record removes the need for such complexity and simplifies NIC TLS drivers.

TOE TLS Transmit

TOE TLS uses a data flow similar to NIC TLS for transmitting TLS records. As with NIC TLS, unencrypted TLS records are passed in the socket buffer directly to the device driver. However, TOE drivers read socket data directly from the socket buffer without relying on software protocols. This simplifies the implementation of TOE TLS compared to NIC TLS.

To probe for TOE TLS, the handler for the `TCP_TXTLS_ENABLE` socket option checks if the current socket is using TOE. If the socket is offloaded, the handler invokes a method on the attached TOE driver to allocate a TLS session.

Similar to NIC TLS, `ktls_frame()` does not set the `M_NOTREADY` flag on TLS mbufs when using TOE TLS. TOE TLS mbufs are inserted into the socket buffer for immediate consumption. TCP invokes the TOE driver output method when data is available. This method reads data from the socket buffer and sends it to the associated NIC's TOE queue for transmit. Since the TOE engine on the NIC is responsible for TCP segmentation and header generation, the TOE method queues complete mbufs from the socket buffer to the NIC. This means that for TOE TLS the method is always able to send complete TLS records in one shot without having to handle edge cases for partial record transmit unlike NIC TLS. In addition, since TOE examines the mbufs in the socket buffer, TOE TLS does not require a separate send tag but uses the TLS session references in TLS mbufs.

TLS Receive

After TLS transmit, kernel TLS receive offload was a natural next step. While web server workloads benefit less from TLS receive offload relative to TLS transmit, other workloads using TLS with more balanced traffic can benefit from TLS receive offload. FreeBSD does not provide an analog to `sendfile()` (such as `recvfile()`) for reading data from a socket, so the main justifications for receive offload are NFS over TLS and supporting TLS receive offload in NICs.

Kernel TLS receive offload uses more invasive changes in both userland and the kernel's socket layer compared to TLS transmit. TLS transmit allows application data to be written directly as larger writes that are split into multiple TLS records by the kernel. TLS receive, on the other hand, returns a single TLS record for each system call. Userland must use `recvmsg()` to read from a socket using KTLS receive as each TLS record is prefixed with a new `TLS_GET_RECORD` control message containing the TLS record header. This permits the userland SSL library to intercept and handle received TLS protocol messages.

The use of control messages for `recvmsg()` requires the receive socket buffer to be treated as a datagram socket holding a linked list of records rather than as a stream socket holding a single linked list of mbufs representing a stream of data. TCP uses an optimized hook for returning received data to userland (`soreceive_stream()`) which assumes a stream socket layout in the socket buffer and does not support control messages. For TLS receive, `soreceive_stream()` was modified to invoke the generic hook (`soreceive_generic()`) if TLS has been

enabled on the socket. Representing decrypted TLS records as datagrams also permits arbitrary mbufs to hold the payload data of TLS records rather than requiring the use of TLS mbufs.

A TLS receive session is created by a `TCP_RXTLS_ENABLE` socket option. This socket option uses the same `struct tls_enable` value used for enabling transmit. For receive this structure was extended to add an initial sequence number field. This is used to handle an edge case where the SSL client library might have read additional data from the socket after the TLS protocol message concluding a key exchange. In that case, the userland library will decrypt TLS records in that pending message in userland. The kernel will only decrypt subsequent TLS records. However, the kernel needs to know the TLS sequence number of the first message it decrypts since the sequence number is used as in input to the authentication phase of TLS record framing. For transmit, the SSL client library invokes the `TCP_TXTLS_ENABLE` socket option before sending any encrypted TLS records, so the initial sequence number for the transmit side is always zero.

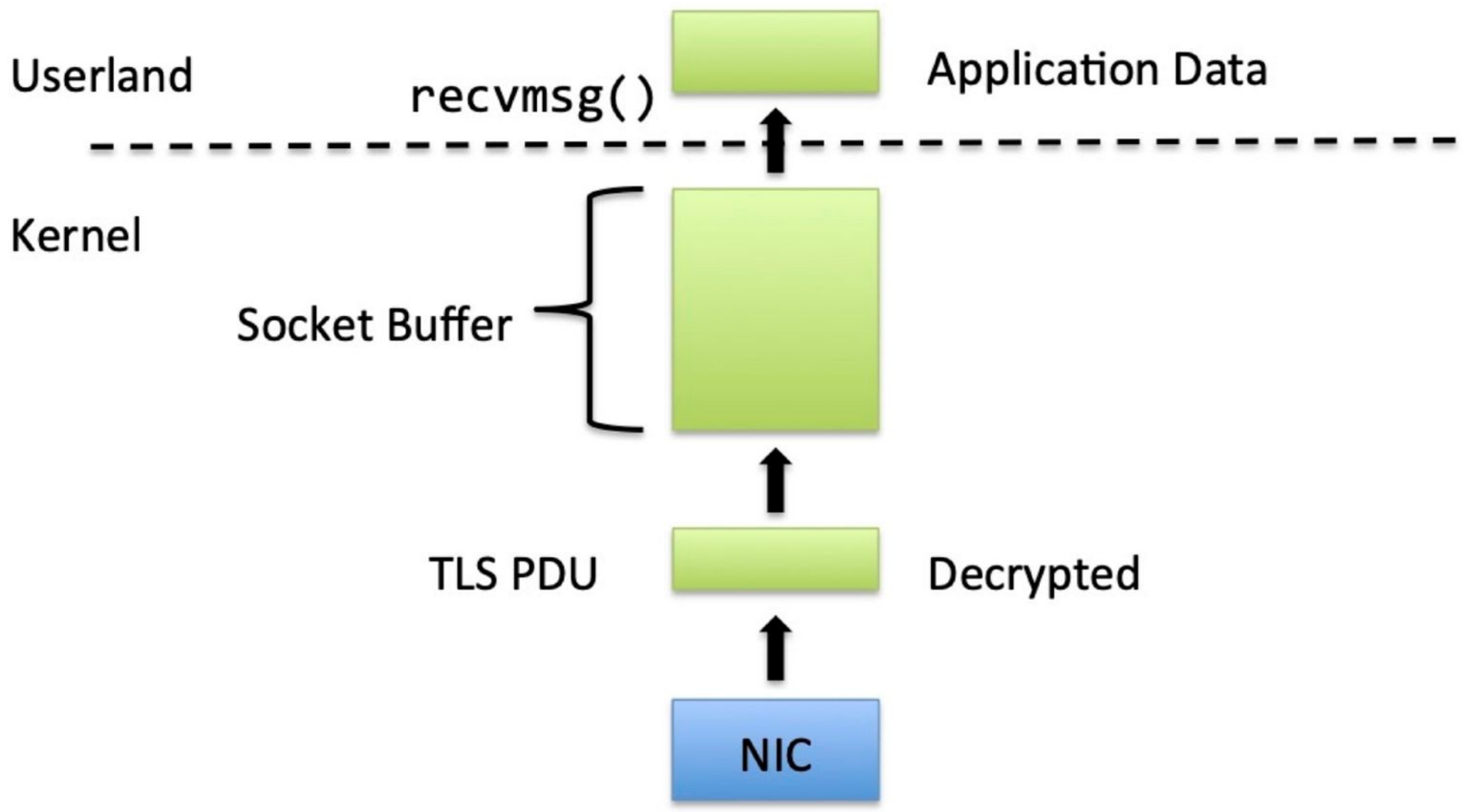
Similar to the transmit case, the handler for `TCP_RXTLS_ENABLE` probes the available backends. If a backend is found, a TLS session is created and associated with the socket's receive buffer. Any data currently present in the socket buffer is treated as encrypted TLS records. The mbufs holding this data are marked as not ready via the `M_NOTREADY` mbuf flag and scheduled for decryption before userland is permitted to read them. If a backend was not found, the `setsockopt()` system call fails and the userland SSL library decrypts TLS records in userland.

Kernel TLS receive currently supports TOE and software TLS backends. NIC TLS backends are not yet supported. For TLS transmit, software TLS was implemented first and NIC and TOE TLS were added later. For TLS receive, TOE TLS was the first backend implemented, and software TLS was implemented second.

TOE TLS Receive

TOE TLS receive uses a straightforward data flow (see Figure 7). The NIC delivers decrypted and verified TLS records to the TOE device driver. The TOE device driver appends these decrypted TLS records directly to the receive socket buffer.

Figure 7: TOE TLS Receive Overview



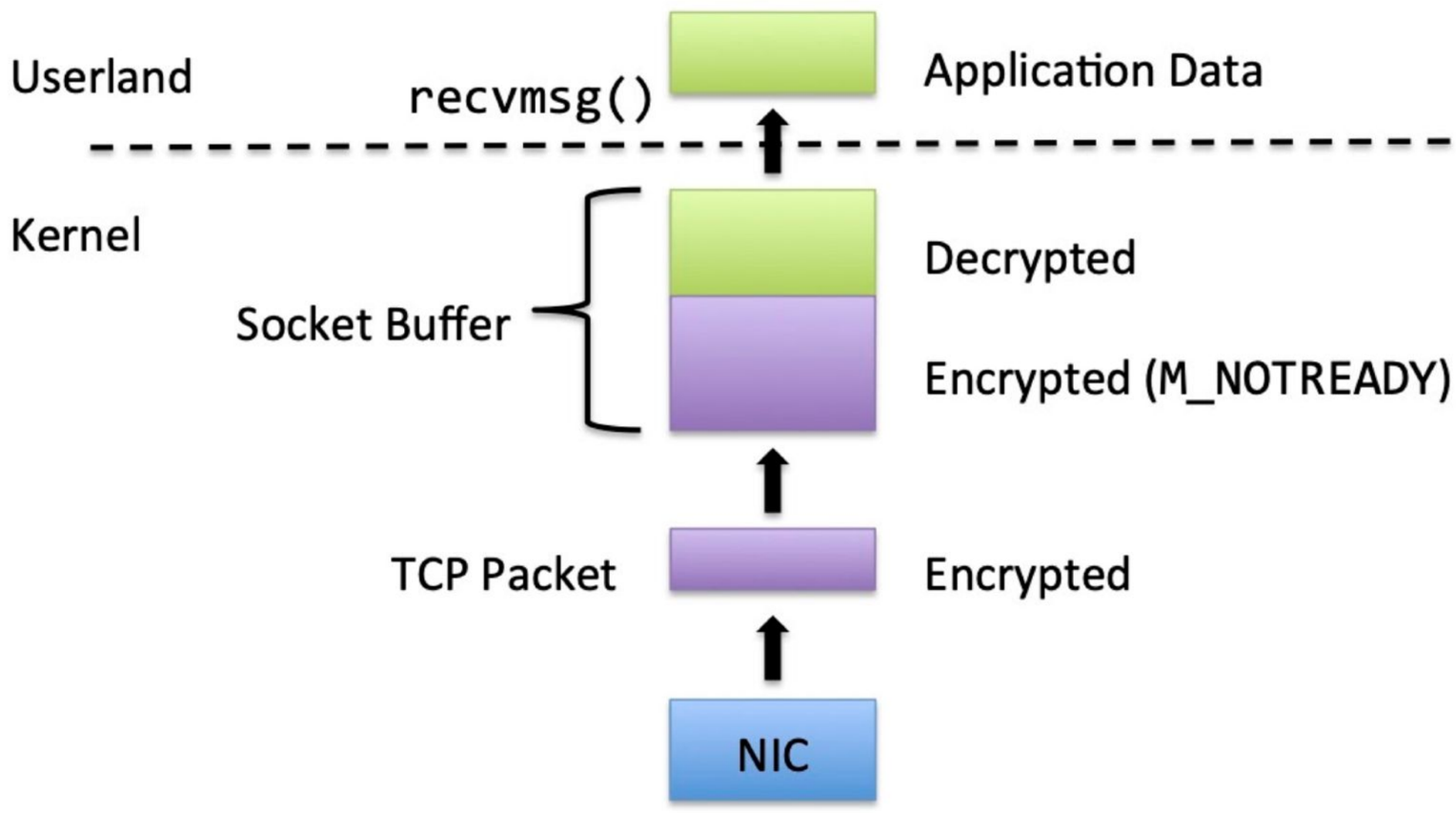
As with TOE TLS transmit, the handler for the `TCP_RXTLS_ENABLE` socket option checks if the current socket is using TOE. If the socket is offloaded, the handler invokes a method on the attached TOE driver to allocate a TLS session.

Appending decrypted TLS records to the socket buffer is straightforward for TOE TLS. The TOE device driver allocates a control message mbuf to hold the TLS header. The driver then passes this control message along with a chain of mbufs holding the TLS record payload to `sbappendcontrol()` to append the decrypted TLS record as a datagram in the socket buffer.

Software TLS Receive

Software TLS receive operates similarly to software TLS transmit, except that the data flows in the reverse direction. Protocols enqueue mbufs containing encrypted TLS records into the socket buffer. These mbufs are marked `M_NOTREADY` until they have been decrypted, at which point they are available for userland applications to read (see Figure 8). As with TLS transmit, decryption of TLS records is performed asynchronously by a pool of worker threads. However, software TLS receive has several differences compared to software TLS transmit.

Figure 8: Software TLS Receive Overview



For TLS transmit, the producer generating data to be stored in the socket buffer knows that it is generating TLS records. The producer in this case being a system call such as `write()` or `sendfile()`. As a result, each transmitted TLS record is stored as a single TLS mbuf. This permits the socket's send buffer to use the normal layout of a stream socket buffer even though it holds a list of TLS records. Software transmit TLS is also able to modify the contents of TLS mbufs in place when encrypting TLS records without modifying the linked list of mbufs in the socket buffer. Once encryption is complete, the `M_NOTREADY` flag is cleared from the mbuf making the TLS record available for transmission by the protocol layer.

For TLS receive, the producer receiving the data into mbufs that are eventually delivered to the socket buffer has no knowledge of TLS. Device drivers receive a stream of packets containing various protocols and belonging to multiple streams. The protocol eventually delivers these mbufs to the socket buffer as a stream of bytes. There is no defined alignment between mbuf boundaries and TLS record boundaries in this case. A single TLS record can span multiple mbufs, and a single mbuf may contain data from multiple TLS records. For example, a TLS record may begin in the middle of one mbuf and continue on through ten or more mbufs before ending. The most natural way to store this stream of mbufs is as a single stream of data as in a normal stream socket. However, the data stored in the receive socket buffer for TLS consists of a list of datagrams that can be read via `recvmsg()`.

Another key difference between software TLS transmit and receive is the lifetime of mbufs in the socket buffer. For both directions of software TLS, the encryption and decryption of TLS records is performed by worker threads. These worker threads must ensure that the mbufs

being modified by a software backend are not freed while the software backend is performing the encryption or decryption. When the transmit side of a socket is closed by `close()` or `shutdown()`, the pending data in the send socket buffer is not immediately freed but is held in the socket buffer until it has been sent to the remote client. When the receive side of a socket is closed, however, the pending data in the receive socket buffer is immediately freed. If one of the worker threads is currently decrypting a TLS record when the receiver side of a socket is closed, TLS receive must ensure that the mbufs are not freed until the decryption has finished.

To handle these differences, TLS receive divides data in the receive socket buffer into three classes. While the socket buffer includes all three types of data in its accounting, each class is stored differently, and accounting details vary by class.

The first class of data stored in a TLS receive socket buffer are decrypted TLS records. These are stored as datagrams in the normal socket buffer chain pointed to by `sb_mb`. All mbufs associated with these TLS records are accounted for the same as in a regular socket buffer including counts of mbufs and mbuf clusters in `sb_ccnt`, `sb_mcnt`, and `sb_mbcnt`.

The second class of data stored in a TLS receive socket buffer are raw protocol data containing encrypted TLS records. These mbufs are stored in a single linked list pointed to by a new `sb_mtls` member. In addition to the regular socket buffer accounting, `sb_tlscc` counts the number of bytes of this class of data stored in the socket buffer.

The final class of data stored in a TLS receive socket buffer are "detached" TLS records. When a worker thread has assembled a chain of mbufs holding an encrypted TLS record, it detaches the mbufs from the `sb_mtls` chain. It then increments a new sb_dtlscc member by the count of bytes in the detached TLS record. After the record has been decrypted, the worker thread checks to see if `sb_dtlscc` has been cleared by a call to `sbflush()` or `sbcut()`. If so, the worker thread frees the mbuf chain. Otherwise, it appends the now-decrypted TLS record as a datagram to the `sb_mb` chain. While a TLS record is detached, the socket buffer accounting does not track the mbufs or clusters used by the TLS record. Only the bytes are accounted for in `sb_tlsdcc` and `sb_ccc`. This still provides an accurate view of the used space to TCP for the purposes of computing the receive window advertised to the remote end of the TCP connection.

Logically, these three classes of data are ordered in the socket buffer as the `sb_mb` chain, followed by any detached TLS record, followed by the `sb_mtls` chain. When `sbcut()` or `sbflush()` walk the socket buffer to free mbufs, they walk the classes in this order to free data. In practice, the only time these functions are invoked on a receive buffer for a TCP socket is to flush the entire buffer when the receive side of a socket is closed. The handling of detached records takes advantage of this by asserting that individual calls to `sbcut()` or `sbflush()` always free all of the detached TLS record or none of it.

When data is appended by the protocol layer to a TLS receive socket buffer, `sbappendstream()` calls a new `sbappend_ktls_rx()` function which appends the new mbufs to the `sb_mtls` chain. After the mbufs have been appended, `ktls_check_rx()` examines the head of the sb_mtls chain. This function reads the TLS header of the next TLS record to decrypt (if a full TLS header is available) and extracts the length field. It then checks if the length of the `sb_mtls` chain against the length from the TLS header. If the full TLS record has been received, `ktls_check_rx()` schedules the socket for decryption by a KTLS worker thread.

KTLS worker threads decrypt TLS records from the `sb_mtls` chain. The worker thread first detaches the TLS record from the receive socket buffer. This removes the mbufs holding the TLS record data from the `sb_mtls` chain but accounts for their data in `sb_tlsdcc`. It then invokes the software backend to decrypt the TLS record. Since the record has been detached, this is performed without holding any socket buffer locks. After the decryption finishes, the

worker thread locks the socket buffer to check if `sbcut()` or `sbflush()` were invoked during decryption. If not, the worker thread allocates a control message to hold the TLS record header, trims the original TLS header and trailer from the original TLS record mbuf chain, and then calls `sbappendcontrol()` to append the TLS record to `sb_mb` as a datagram.

Final Thoughts

Kernel TLS is an exciting, yet complex new feature in FreeBSD 13.0. At the time of writing, support for kernel TLS transmit has been upstreamed to FreeBSD's head branch including support for software TLS, NIC TLS, and TOE TLS. A ktls_ocf kernel module provides software TLS support for AES-GCM cipher suites for connections using TLS protocols 1.2 and 1.3. Since this module uses the kernel's opencrypto framework, it permits TLS records to be encrypted by crypto co-processor drivers and not just via software run on the host CPU. Chelsio and Mellanox device drivers both include support for NIC TLS, and Chelsio's TOE device driver includes support for TOE TLS. On the receive side, the core TLS receive framework including support for TOE TLS has been merged to the head branch. Software TLS receive is still in review but should appear in head prior to the release of 13.0. Ongoing work continues to further optimize performance as well.

Kernel TLS also requires changes to userland SSL libraries. At the time of writing, OpenSSL's development branch includes support for TLS transmit for TLS 1.0-1.2 on FreeBSD. This functionality is available as a KTLS option in the security/openssl-devel port. In addition, the current KTLS patches have been backported to OpenSSL 1.1.1 as a KTLS option for the security/openssl port. Support for TLS transmit for TLS 1.3 and TLS receive for TLS 1.1-1.2 is currently in review and should land in OpenSSL prior to the release of 3.0.0.

Netflix also developed an extension to nginx to take advantage of sendfile() over TLS using the new SSL_sendfile() function in OpenSSL. This is currently available as a patch to the www/nginx-devel port under a KTLS port option.

Finally, kernel TLS is a large project that has been worked on by several members of the FreeBSD community. The current work is the product of iterations and refinements over multiple years. Scott Long first envisioned moving TLS into the kernel while working at Netflix. He worked with Randall Stewart (also at Netflix) to design and implement the first versions of software TLS transmit. Drew Gallatin introduced external pages mbufs and their later extension as TLS mbufs and converted Netflix's early KTLS to use `M_NOTREADY` mbufs for software TLS transmit. Drew also added a pluggable interface for software TLS backends. I worked with Drew to add the infrastructure for NIC TLS transmit as well as support for NIC TLS transmit on Chelsio T6 adapters. Hans Petter Selasky added support for NIC TLS on Mellanox ConnectX-6 Dx adapters. I subsequently worked on TOE TLS and TLS receive. Scott's, Randall's, and Drew's work was funded by Netflix. Hans Petter's work was funded by Mellanox. My work has been funded by both Chelsio and Netflix.

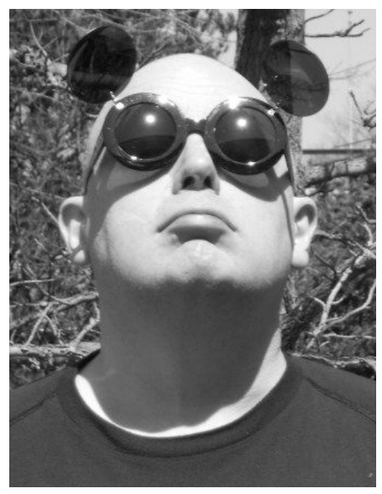
JOHN BALDWIN is a systems software developer. He has directly committed changes to the FreeBSD operating system for 20 years across various parts of the kernel (including x86 platform support, SMP, various device drivers, and the virtual memory subsystem) and userspace programs. In addition to writing code, John has served on the FreeBSD core and release engineering teams. He has also contributed to the GDB debugger and LLVM. John lives in Concord, California, with his wife, Kimberly, and three children: Janelle, Evan, and Bella.

AN • INTERVIEW • WITH

Michael W Lucas

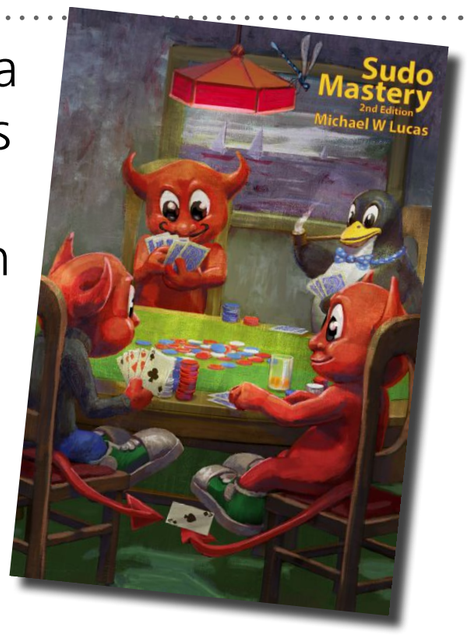
by Allan Jude and Benedict Reuschling

This is excerpted from BSDNow, Episode 329, recorded December 2019. Allan Jude and Benedict Reuschling interview Michael W Lucas.



BENEDICT REUSCHLING: Michael W Lucas has been interviewed on this show a couple of times. His last interview was a few months ago, and we thought you might like to know what he has been doing and what he has planned. Here's the interview. Enjoy!

REUSCHLING: Welcome, Michael! You released a new book recently that I'm sure listeners would like to hear about. Can you tell us a bit about that book? It's about sudo, I hear.



MICHAEL W LUCAS: Well, thank you for having me. And yes, the new book is a second edition of *Sudo Mastery*. Sudo hasn't undergone extensive changes unless you're using LDAP. But it has undergone a lot of minor changes, so, basically, this second edition is about catching up on that and making sure that the information is accurate for today's sudo. I also dove deeply into some changes in the recommended configuration. For example, sudo has features like checksum verification to make sure binary hasn't been tampered with. And, of course, nobody is going to go and compute to checksum of every binary by hand, let alone update them whenever they're patched. So, I did some scripting for popular operating systems to programmatically compute the checksum for everything.

REUSCHLING: Oh, that's useful.

LUCAS: It can be. If you suspect your system is vulnerable to outside tampering, checksumming may be a solution for you. And there are some new calls out there. They used to have a tool to transform a sudoer's file into LDAP format. That was handy and useful, but now they have a general tool to transform sudoers to and from LDAP and then to and from JSON. So, you can put your sudoers in any format you want. You can feed it to your auditing program and make sure that everything is the way you want. And, of course, the book has a beautiful cover.

JUDE: It does! Will you explain the different versions of that wonderful cover?

LUCAS: If you buy the e-book, you get a very nice cover based on one of the classic "dogs playing poker" image. And since people have asked, yes, there's a reason one of the BSDs is wearing a black turtleneck. And the reason another BSD is slipping him an ace.

If you get the paperback, you'll get a more complete version of that, and hardcover wraps around on the cover. If you have the hardback, the dust jacket has art continued onto the inside flap. So, if you're looking for something fun and pretty, this book will do it. I'm really pleased with how it came out.

JUDE: So, you have cover art that comes in bigger and bigger and bigger sizes. What are you going to do next?

LUCAS: I've been pondering what comes next. I'm not exactly at a loss because I have lots of ideas. But I just finished a really huge project with the jails book [*FreeBSD Mastery Jails*]. I had to write six books to write the jails book. It was a giant project, and everything was aimed at a particular destination. Plus, I wrote a couple of little books on the side as I was treading through it. I no longer have a master plan, but I'm trying to come up with an overarching goal. And whatever I do in terms of art, books, what have you, will feed into that. So, I do have some ideas, and thanks, but I'm not asking the whole Internet to send me suggestions.

REUSCHLING: It might be too late now!

JUDE: Speaking of the jails book, almost everyone who uses FreeBSD could learn quite a bit from it, even people like me who have been using jails and FreeBSD for years. There's more stuff in the book than in my head! Were there surprising things you learned while writing it?

LUCAS: I've been using FreeBSD since late 1995 and doing that work forced me to go back and review everything I knew—and some things have changed since 1995. I learned a lot of little edge cases on filesystems. You can really do some amazing things with DEVD, and even nullfs is useful. It's amazing how fine-grained some of the controls are in jails now. I remember when System 5 IPC in a jail was simply not a thing, and now you can turn it on and off per jail and even turn on System 5 IPC features individually per jail. That's something I know was implemented because people needed it, but I'm sort of afraid to find out why someone needed System 5 shared memory—but not semaphores.

JUDE: Yeah, I think part of that was back in the past when the IPC name space was all one—you maybe want a jail to be able to only see parts of it. Although a number of years ago when we finally got namespacing there so you could have Postgres in two different jails that wouldn't accidentally stomp on each other, it maybe became less of an issue.

LUCAS: I thought that at first, but that feature evolved later. So, this was some sort of design decision. I trolled through 20 years of history, and the order in which things happened was really illuminating. It's a great example of iterative development. I mean jails saved me a lot of pain back in 1999, 2000, and they've just crept forward over the decades. Jails are hard is the short answer. They are the culmination of everything in systems administration. You must know filesystems. You must know upgrades. If you have a lot of jails, you really have to have some sort of orchestration or automation. This stuff gets very complicated very quickly because all of a sudden you could run 500 virtual machines on one host. You have to manage 500 virtual machines on one host. And that's just terrible. Nobody wants to do that.

REUSCHLING: Many people will end up grabbing the book when they are in need of a jail.

LUCAS: Yeah. It's out there. People who need it will find it. People who don't need it, well, they have a simpler life than the rest of us.

JUDE: Oh, you know, I've gone overboard with the things you can do when you combine ZedFS and jails to the point where I needed a system where customers need to be able to upload files but need to make sure they can only upload their own files and so on.

LUCAS: Yes. Anything like that I would certainly put in a jail. Anything that faces the public I want to put in a jail.

JUDE: Yeah, it's really interesting how well ZedFS and jails fit together, considering, on one hand, that Sun, when they developed ZedFS, integrated it with zones, which are kind of like jails. But they're also quite a bit different, and the fact that we've managed to make it line up so well I find really pleasing.

LUCAS: It is nice, and I think the issues that apply to getting a virtualization aware of a filesystem are not unique to zones. They simply set up a filesystem that can be delegated and that can have lower-level permissions.

JUDE: It's really interesting that you can do both. You can jail a dataset, which basically means you're delegating it to root in the jail. But then root in the jail can delegate to an individual user in that jail.

LUCAS: It really is lovely.

JUDE: Um-hum.
REUSCHLING: There are use cases that you typically don't think about at the beginning, but then you're like, yeah, it's possible and I can see a good application for it.

LUCAS: Yeah, really, systems administration is about looking at the tools you've got and figuring out how to plug them together to solve a problem. And ZedFS is a great tool.

JUDE: So, moving on, how did you come to the realization that the world was in need of a book about Simple Network Management Protocol—SNMP?

LUCAS: Oooohhhh. Because it's everywhere and none of the existing books thrilled me. And SNMP is really not as hard as you might think.

REUSCHLING: Well, that's what the S is for, right?

LUCAS: Yes, yes. And the truth is that the actual protocol is very simple. It has seven parts—that's it. The problem is with what all of the vendors and implementers have done with those seven parts. And no matter how specifically you write the standard, there is always room for interpretation. What we need is not a book on how glorious it is, but a book on "these are the pieces, here is where we are, and here is how you figure out this stuff."

When it works, it's fantastic. People use SNMP to monitor all the time. SNMP is well-suited for managing networks. Even today in talking to people and doing the research, there are folks who say if you have a really saturated Internet link, you may need to issue a command to a remote device to solve the problem. We have SNMPv3, which a lot of people are scared of. After digging into it, really, I think we're just teaching it backwards. Sometimes it is the only solution you have, and I would not suggest that you build your whole network around SNMP. But I would say it's a tool you need to know whether you're a network admin or a sysadmin. Sometimes you need that particular tool, or a tool will solve things that are really hard to solve in other ways. There's so much confusion about it.

JUDE: Yeah, you know, it's definitely a tool in my toolbox, and I am not shy to admit that I didn't ever really learn it properly. I just kind of muddled through enough to get what I needed out of it.

LUCAS: Yeah, and you can discover the information you need in two packets.

JUDE: Right. When I am running that every 30 seconds, it's a lot nicer than having something much more heavyweight.

LUCAS: Mind you, I have learned things like you can run SNMP queries over SSH.

JUDE: Ohhh!

LUCAS: Which I'm not covering because the book is already too big. But it is a possible solution for some sorts of problems. If you have a certificate authority infrastructure already, you can run SNMP over TLS.

JUDE: Does that still go over UDP or does it change to TCP in that case?

LUCAS: It can use either. DTLS is TLS over UDP.

JUDE: Yeah, for datagrams.

LUCAS: There are a lot of options there. If I can successfully orient people so they can separate learning the protocol from learning a vendor's weird configuration, it'll be more than worth it.

REUSCHLING: What else did you learn about SNMP during the research for the book?

LUCAS: Well, everything is terrible, but that's not a surprise. Let's see. A lot of people have done interesting things with it. There are agents from MYSQL and Postgres and Apache that will plug right into net SNMP. So, you can easily pull statistics from all sorts of software. There is extending it. You can have SNMP run arbitrary commands on your server. Which may or may not be a good thing depending on how complicated you want to be.

JUDE: And graph trends and all kinds of stuff.

LUCAS: Oh, yes, yes. Of course, the problem with monitoring things like that is you don't need to monitor them until they go horribly wrong—I love system administration!

JUDE: Looking back, how did your planning for the year work out?

LUCAS: Well, I had all sorts of things I wanted to do. About a year ago, they took out half my thyroid, and all of the mysterious health issues I've had for the past few years cleared up. And this is great! This is fantastic! It meant I could do everything. And I had to learn once again the hard way that I can do things, but I can't do everything. I planned to write a bunch more words than I did. I accepted too many invitations to travel, and travel messes me up. So, I've decided to do less travel and more books. I make words for a living. This is what I've always wanted to do. I'm very lucky to be able to, and I just have to stand here at my keyboard and do the work.

JUDE: Well then, speaking of that, can you tell us what you're planning?

LUCAS: I've got a couple of thoughts on tech books. The DNSSEC book is due for an update. And with DNS over HTTPS becoming more of a thing, that should probably go in there as well.

JUDE: Especially now that we're in an age where most people are using it and don't understand this stuff.

LUCAS: Yes.

JUDE: People are making an informed decision to use DOH.

LUCAS: Right. It's a great immediate solution for people living under censorship regimes. However, authoritarian and censorship states don't put up with those kinds of problems for very long.

JUDE: Yeah, and I think the most dangerous part is that it's not the whole solution. If you depend only on it, you're going to run into problems.

LUCAS: Yes, yes, quite a few problems. But I can do a book on how to set it up. And email could be an interesting book. There's really nothing on how to set up a complete email solution these days. So, that's a possibility.

JUDE: Yeah, doing DKIM wasn't too bad, but the newer DMARC stuff was very confusing to me.

LUCAS: I think email privacy is mostly lost. Even if you run your own mail server, chances are you are communicating with someone who uses one of the big commercial mail companies. But it's still worthwhile trying to run your own mail servers, and that's a topic I've been looking at. And I'm starting to ponder a couple of OpenBSD things—and anytime I mention writing an OpenBSD book, people jump up and want the next Absolute OpenBSD. Let me say there are no definite dates for that—I'm glad you want it, and it will come.

JUDE: What about the sequel to *git commit murder*?

LUCAS: That is on my list. I tried writing it while I was very ill. Right now, I'm pushing to get the SNMP book done. The nonfiction brings in the money, and I'm glad I have it. I'm not knocking it, but I do not have enough novels out to make a living by writing *git commit* books. I will be very happy when I'm finished with SNMP.

REUSCHLING: Any final thought today, Michael?

LUCAS: I say thank you to all the people who buy the books and that I'm glad you find them helpful. Also, if you've managed to listen to this interview and me babbling on this long, thank you for that too.

REUSCHLING: Thank you, Michael, for your continued writing and your books.

P.S. Today, Lucas finds his statement about traveling less disturbingly prescient.

Thank you!

The FreesBSD Foundation would like to acknowledge the following companies for their continued support of the Project. Because of generous donations such as these we are able to continue moving the Project forward.

FreeBSD™ FOUNDATION

Are you a fan of FreeBSD? Help us give back to the Project and donate today! **freebsdfoundation.org/donate/**

Please check out the full list of generous community investors at freebsdfoundation.org/donors/

Uranium

intel

Iridium

NetApp

Platinum

NETFLIX

Gold

facebook JUNIPER NETWORKS

Silver

BECKHOFF Microsoft STORMSHIELD Tarsnap vmware

FreeBSD® JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!

2020 Editorial Calendar

- FreeBSD in Research (Jan/Feb 2020)
- Filesystems (March/April 2020)
- Network Performance (May/June 2020)
- Benchmarking/Tuning (July/August 2020)
- Contributing/Onboarding (Sept/Oct 2020)
- Workflows (Nov/Dec 2020)

Find out more at: freebsdfoundation/journal

FreeBSD® guides

FreeBSD Desktop Distributions

BY DREW GURKOWSKI

FreeBSD desktop distributions have taken the FreeBSD operating system in a new direction while still relying on the incredibly powerful and stable base operating system of FreeBSD. From integrated desktop environments, portable USB-based systems, to data recovery tools, FreeBSD has been used by a wide variety of open-source projects.

FuryBSD

Novice to Advanced Users

Project Website

"FuryBSD is a brand-new, open-source, FreeBSD desktop. FuryBSD pays homage to desktop BSD projects of the past like PC-BSD and TrueOS with its graphical interface and adds additional tools like a live, hybrid USB / DVD image."

What Makes FuryBSD Unique?

System Attributes:

- A deployment vessel including:
  - Stock FreeBSD,
  - FreeBSD-update, pkg, bsdconfig, bsdinstall,
  - Firefox,
  - XFCE desktop environment.
- Small 1.8GB download size.
- Adds a live, hybrid USB / DVD image to stock FreeBSD.
- A read write filesystem that verifies the hardware works before committing to disk.
- Minimizes disk commitment.
- Automatically installs a graphical desktop environment.
- Pristine copy of FreeBSD, allowing the system to continue updating FreeBSD tools.
- Allows for the configuration of devices for mass deployments. Automation with possible configuration for each step.
- Latest patchset is updated quarterly, so users don't have to update using FreeBSD-update.
  - The build system uses Poudriere image to ensure the jail is up-to-date.
- Offers .iso media including RELEASE + FreeBSD-update patches.
- If FuryBSD depreciates, system can still be used by updating FreeBSD.

User-end Attributes:

- Boots the user directly to desktop of choice, connects to network automatically (Ethernet), or provides a tool to make connecting to WiFi easier.
  - Minimizes the need for a potentially confusing setup process for novice users.
- Includes an Xorg tool to allow users to try various driver combinations without having to install each one.

    - Allows for various combinations of drivers without having to test each one.
- Every user change is saved unless the system is rebooted without installing.
- Recently released their own forum for FuryBSD users.

Notable Future Plans:

- Creating more documentation for the project, focused on how to get started and troubleshooting any possible issues.
- Out-of-box printing support.
- A way to load 3rd-party drivers and wireless adaptors.
- More default applications to provide a broader desktop experience.
- Continuous integration for application updates.
- A custom pkg repo.

MidnightBSD

Novice to Advanced Users

Project Website

"MidnightBSD is a BSD-derived operating system developed with desktop users in mind. It includes all the software you'd expect for your daily tasks—email, web browsing, word processing, gaming, and much more."

What Makes MidnightBSD Unique?

System Attributes:

- New window and login manager.
- Initially forked from FreeBSD 6.1 beta.
    - To allow customization of ports and system configuration.
    - Includes syncs from later FreeBSD versions.
- Fully integrated environment with a focus on stability and optimization for desktop users.
- Scheduling, allocation, security, and app support are uniquely tailored for desktop users.
- Less emphasis on server optimization.
- Includes a range of tools and utilities, including:
    - ZFS,
    - Hast,
    - OpenSSH,
    - Perl.
- Imported features from DragonFlyBSD/OpenBSD/NetBSD.
- Improved security with ipfw and sudo.

User-end Attributes:

- New window and login manager.
- Centralized system preferences.
- Emphasis on easy-to-use graphical interfaces for new users.
- BSD-styled command line.
- Graphical ports and package manager.

- Large user community, including mailing lists and community forums.
- Extensive documentation.
- Bugzilla for troubleshooting.

Notable Future Plans:

- Expanded security capabilities.
- Rewriting the package cluster system.
- Maintaining and updating ports.

GhostBSD

Novice to Advanced Users

Project Website

"GhostBSD is a simple, user-friendly operating system, based on the legendary security and stability of the FreeBSD operating system. GhostBSD provides all the benefits of the FreeBSD operating system combined with a focus on simplicity for newcomers."

What Makes GhostBSD Unique?

System Attributes:

- Maintains the stability and security of FreeBSD.
- Optimized for modern workstations, desktop setups.
- Massively simplified.
- Free and open with source code on GitHub.
- Optimized codecs for multimedia files.
- Project release as well as community releases.
- A GhostBSD package repository with 30,000+ applications.
  - Slow-moving rolling release.
- Can be updated by command line.
- Switched to TrueOS in 2018.
  - Built from 12.0-STABLE.
  - TrueOS depreciated in 2020.

User-end Attributes:

- Shipped with a simplified installer.
- Choice between desktop environments.
- Comes with preinstalled commonly used software.
- Robust community controlled wiki.
- Easy to contribute to the project/report bugs.
- Community has full control of the GhostBSD community releases.
- Community forums for GhostBSD and FreeBSD questions/discussion.

Notable Future Plans:

- Possibility of a community release around the plasma5 desktop.
- Movement toward a rolling release schedule.
- Post TrueOS development.

NomadBSD

Novice to Advanced Users

Project Website

"NomadBSD is a persistent live system for USB flash drives, based on FreeBSD. Together with automatic hardware detection and setup, it is configured to be used as a desktop system that works out of the box, but can also be used for data recovery, for educational purposes, or to test FreeBSD's hardware compatibility."

What Makes NomadBSD Unique?

System Attributes:

- Extremely low system requirements.
  - Can be run by computers with a 1.2GHz CPU and 1GB of RAM.
- A fully integrated desktop environment optimized by the operating system.
- Further executable programs with some settings adjustments.
- Requires a more advanced USB, USB3 preferred.
- The desktop environment is curated specifically for the OS.
- Tons of support for using the OS as a test environment.
- Minimal size while including a curated collection of applications.
- Automatic hardware detection and setup.
- Because of the boot process, can be used for data recovery.
- Small development team, more unified vision.
- Uses a live USB, allowing changes to be saved to a portable system.

End-user Attributes:

- Automated, simplified setup with limited device configuration.
- Can serve as a tool for FreeBSD developers. (testing/data recovery).
- Focus on easy-to-use, guided computing.
- Innovative desktop features.
- New community forums created in March 2020.
- Extremely portable due to size/live USB setup.

DREW GURKOWSKI, FreeBSD Foundation

FreeBSD® guides

Bulk Ports Management with Poudriere

BY DREW GURKOWSKI

Step 1: Using the Ports Collection to Install Poudriere

All commands in this section should be run as root; using the `su(1)` command will work. To download a compressed snapshot of the Ports Collection into `/var/db/portsnap`:

```
# portsnap fetch
```

When running `portsnap` for the first time, extract the snapshot into `/usr/ports`:

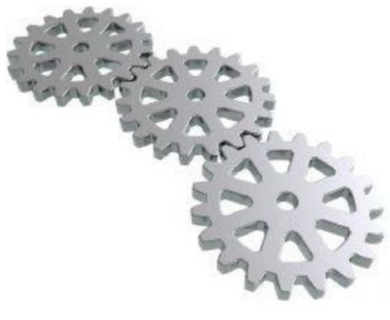
```
# portsnap extract
```

Next, build and install Poudriere from the Ports Collection using:

```
# cd /usr/ports/ports-mgmt/poudriere
# make install clean
```

Prompts will appear throughout the installation process, stick with the default settings and install the port.

Step 2: Configuring Poudriere



Poudriere is an extremely powerful tool designed for package production, but it can also be used to bulk manage ports. In order to do so, a few small tweaks will need to be made to the configuration before moving forward.

First, copy and move the configuration file to the correct location with:

```
# cd /usr/local/etc
# cp poudriere.conf.sample poudriere conf
```

Then use the `ee(1)` text editor to edit the copied configuration file:

```
# ee poudriere.conf
```

Arrow keys can be used to navigate down to the line:

```
FREEBSD_HOST=_PROTO_://_CHANGE_THIS_
```

Edit the above line, so it instead looks like this: (backspace to delete text)

```
FREEBSD_HOST=ftp://ftp.freebsd.org
```

Type `ESCAPE` and then `ENTER` twice to exit and save changes to the configuration file.

Step 3: Set Up a Poudriere Jail

Before continuing, Poudriere will need to fetch and extract its own version of the FreeBSD Ports Collection. In order to do this, run:

```
# rehash
# poudriere ports -c
```

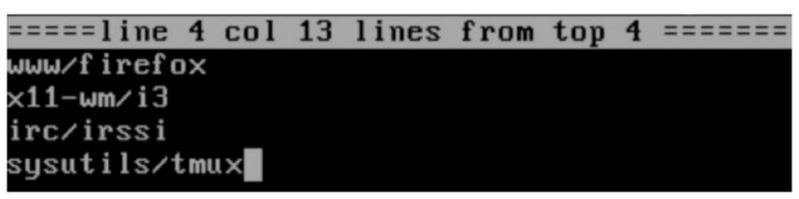
A FreeBSD jail will need to be set up for Poudriere to perform bulk functions. This can be done with the commands:

```
# mkdir /usr/local/poudriere
# poudriere jail -c -j 91x64 -v 12.1-RELEASE -a amd64
```

In the above command, `91x64` identifies the jail, while `12.1-RELEASE` identifies the FreeBSD version to use. These can be tweaked if a different name or FreeBSD version is required. Just remember to also replace them in the rest of this guide.

Step 4: Creating a Bulk List of Ports

Before continuing, run the following:




```
# cd poudriere.d
# echo WITH_PKGNG=YES >> 91x64-make.conf
```

The next step will be to create a list of ports for Poudriere to compile and maintain; a wide range of ports is available in the FreeBSD Ports Collection, and Poudriere can be used to manage them all. Start with:

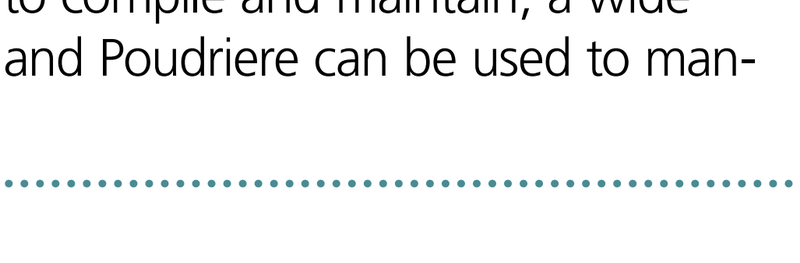
```
# cd /usr/local/etc
# ee poudriere-list
```

As before, use the `ee(1)` text editor to edit the file, adding a list of ports for Poudriere to manage. Above is an example image including Firefox, the i3 Windows manager, irssi, and tmux. Ports can be added to the list by using the port origin (category/name).

Type `ESCAPE` and then `ENTER` twice to exit and save changes to the configuration file.

Step 5: Configuring Poudriere Installation Options

This step is optional unless manual configuration is needed.


tmux-3.0a_1
[x] BACKSPACE Build with tty/keys patch
[x] DOCS Build and/or install documentation
[x] EXAMPLES Build and/or install examples
[ ] LIBEVENT_STATIC Build with static libevent
[ ] UTF8PROC Build with utf8proc support
< OK >
<Cancel>

While Poudriere can be used to automate bulk port management, it still allows users to manually configure each port. However, this step can be done before installation instead of having to

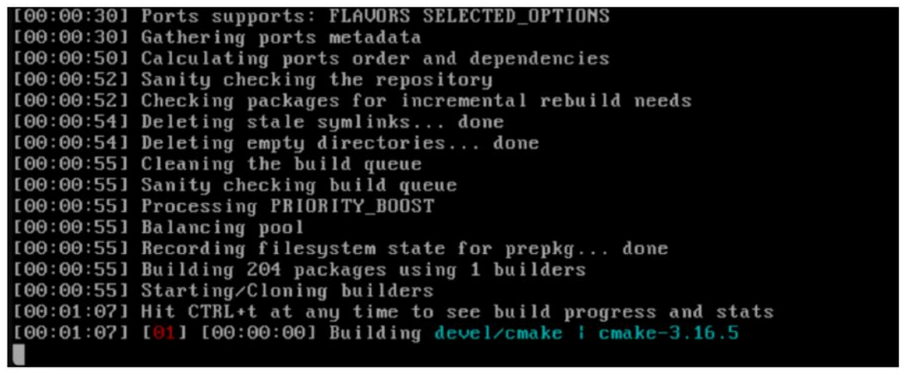
be present for the entirety of the process. If, for instance, the user wants to edit `tmux(1)`, use the command:

```
# poudriere options  -c sysutils/tmux
```

Then, using the installation prompts (arrows keys to navigate, space to select) the port can be manually edited. This process can be done for each port that Poudriere manages.

Step 6: Using Poudriere to Manage Ports

With configuration complete, installing the entire list of ports can be done with just one command:


[00:00:30] Ports supports: FLAVORS SELECTED_OPTIONS
[00:00:30] Gathering ports metadata
[00:00:50] Calculating ports order and dependencies
[00:00:52] Sanity checking the repository
[00:00:52] Checking packages for incremental rebuild needs
[00:00:54] Deleting stale symlinks... done
[00:00:54] Deleting empty directories... done
[00:00:55] Cleaning the build queue
[00:00:55] Sanity checking build queue
[00:00:55] Processing PRIORITY_BOOST
[00:00:55] Balancing pool
[00:00:55] Recording filesystem state for prepkg... done
[00:00:55] Building 204 packages using 1 builders
[00:00:55] Starting/Cloning builders
[00:01:07] Hit CTRL+t at any time to see build progress and stats
[00:01:07] [01] [00:00:00] Building devel/cmake | cmake-3.16.5

```
# poudriere bulk -j 91x64 -f poudriere-list
```

Poudriere will take some time to complete the bulk process, but unlike manually building and installing ports, it can do all the work without needing user inputs. Poudriere also offers a text-based visual display of the installation process. It will allow users to have greater control over their ports without sacrificing automation.

If an update is needed to the port list, Poudriere can update the Ports Collection with the commands:

```
# cd /usr/local/etc
# poudriere bulk -j 91x64 -f poudriere-list
```

DREW GURKOWSKI, FreeBSD Foundation

Jails ARE FBSD'S MOST LEGENDARY FEATURE:

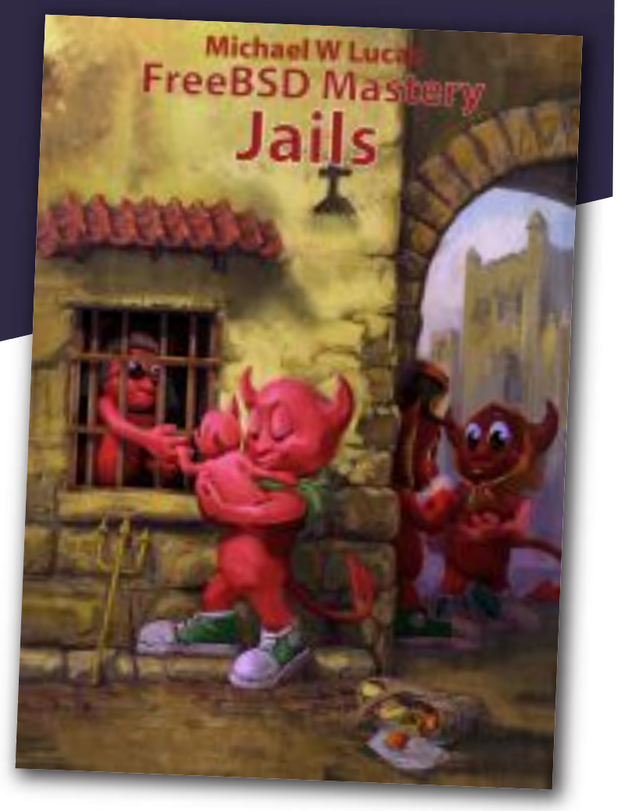
KNOWN TO BE POWERFUL, TRICKY TO MASTER, AND CLOAKED IN DECADES OF DUBIOUS LORE.

FreeBSD Mastery: Jails cuts through the clutter to expose the inner mechanisms of jails and unleash their power in your service.

Confine Your Software!

- Understand how jails achieve lightweight virtualization
- Understand the base system's jail tools and the iocage toolkit
- Optimally configure hardware
- Manage jails from the host and from within the jail
- Optimize disk space usage to support thousands of jails
- Comfortably work within the limits of jails
- Implement fine-grained control of jail features
- Build virtual networks
- Deploy hierarchical jails
- Constrain jail resource usage.

...And much, much more!



FreeBSD Mastery: Jails BY MICHAEL W LUCAS ***Available at All Bookstores***

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.

freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

Events Calendar

BSD Events taking place through August 2020

BY ANNE DICKISON

FreeBSD Fridays: A Series of 101 Classes

Fridays, beginning July 17, 10 a.m. PDT

VIRTUAL

Beginning July 17, please join us for an expanded series of 101 classes. We're building on our FreeBSD Day content to offer 1-hour live sessions designed to get you started with FreeBSD. The program kicks off with Deb Goodkin's *Intro to FreeBSD* class on July 17, followed by a two-part *Installfest* led by Roller Angel on July 24 and July 31. Other classes include *A Brief History of the BSD Fast Filesystem* by Kirk McKusick on August 7 and *Introduction to OpenZFS* by Dan Langille on August 14. Stay tuned for more information! All sessions are at 10 a.m. PDT.

FreeBSD Office Hours

Bi-Weekly Wednesdays at 18:00 UTC

https://wiki.freebsd.org/OfficeHours

Join members of the FreeBSD community every other Wednesday for FreeBSD Office Hours. From general Q&A to topic-based demos and tutorials, Office Hours is a great way to get answers to your FreeBSD-related questions. Past episodes can be found at the FreeBSD YouTube Channel.

Write For Us!

Contact Jim Maurer with your article ideas.

(jmaurer@freebsdjournal.com)